1992

On The Mapping of Partial Differential Equation Computations onto Distributed Memory MIMD Parallel Machines (Ph.D. Thesis)

Nikos Chrisochoides

Report Number:
92-101

ON THE MAPPING OF PARTIAL DIFFERENTIAL EQUATION COMPUTATIONS ONTO DISTRIBUTED MEMORY MIMD PARALLEL MACHINES

Nikos Chrisochoides

CSD-TR-92-101
August 1992

ON THE MAPPING OF PARTIAL DIFFERENTIAL EQUATION COMPUTATIONS ONTO DISTRIBUTED MEMORY MIMD PARALLEL MACHINES

A Thesis

Submitted to the Faculty

of

Purdue University

by

Nikos Chrisochoides

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 1992

This thesis is dedicated to the memory of my father and to my mother; without her sacrifices my education will never have been succesful.

ACKNOWLEDGMENTS

The learning process requires constant and persistent effort. This thesis is the result of such an effort; it is the result of a generously given patience, moral support and friendship of many people. I acknowledge collectively all of them, whether they are mentioned below or not.

My teacher and advisor Elias Houstis provided guidance and patience during the first difficult years of my graduate studies. For this, for his constant guidance and support and for the opportunity he gave me to continue my studies towards a Ph.D at Purdue University, I am grateful.

I am thankful to John Rice who as a co-advisor guided me with his insights and intuition. Also, I am thankful to both John Rice and Elias Houstis who as principal investigators of the Parallel Ellpack project created the best possible environment for research.

Equally, I thank the members of my commitee Dan Marinescu, Aditya Mathur and George Vanecek for giving their comments on my thesis.

I am thankful to Apostolos Hadjidimos, Rober Lynch, Walter Gautschi and Mokhtar Aboelaze for showing me the elegance and beauty of topics from Matrix Analysis and Orthogonal Polynomial to Systolic Algorithms.

I am also thankful to Panos Papachiou, Cathrine Houstis, Stavros Kortesis, Bill Bouma, and Chandrejit Bajaj for the productive research discussions and collaboration I had with them.

My discussions through my graduate years with Scott McFaddin and the collaboration with all the members of the Parallel Ellpack group was a useful experience, I am thankful to all.

Also, I thank the Computer Science Department for the economic support during my graduate years at Purdue University, the Department of Energy for supporting me during the summer school at Argone National Lab, the NSF which through Gene Golub, Ahmed Sameh, and Apostolos Gerasoulis generously supported me in attending the parallel circus workshops.

My brother George Chrisochoides contributed much in this thesis by teaching me, during the early years of the elementary and high school, the art of formulating problems and methodologies that required the composition of more than a few simple true statements. Also, I am grateful to my family in Greece and my mother, father, sisters and brother in law as well as to my friend Theodore Dendroulakis for their love, friendship and hospitality during all these years.

Finally, I am grateful to my best friend, wife and colleague Pelayia Varodoglu who as an inexhaustible and persistent source of love, friendship, support and patience was the invisible helm that helped me to start and finish this thesis. Also, I thank her for the proofreading of the whole thesis.

TABLE OF CONTENTS

Page

LIST OF TABLES

Figure Page

LIST OF FIGURES

ABSTRACT

Chrisochoides, Nikos. Ph.D., Purdue University, August 1992. On the mapping of Partial Differential Equation computations onto distributed memory MIMD parallel machines. Major Professor: Elias Houstis.

The mapping of the computations associated with both matrix and domain decomposition methods for the numerical solution of Partial Differential Equations (PDEs) into load balanced tasks requiring minimum synchronization and communication is a difficult combinatorial optimization problem and its optimal solution is essential for the parallel processing of PDE computations. Often determining data mappings that optimize a number of criteria, like workload balance, synchronization and local communication involves the solution of a NP-Complete problem.

This thesis deals with the automatic mapping strategies for the computations associated with the numerical solution of PDEs onto distributed memory MIMD machines. In the case of PDE computations, such mappings can be formulated at three distinct levels: the discrete geometrical data structures associated with the PDE domain, the linear system of algebraic equations associated with some discretization of the PDE equations, and the data flow graph of the PDE solver. In this thesis we formulate and analyze mapping strategies based on the first two levels.

In the geometry mapping strategies we formulate the mapping problem at the discrete geometric data structures (element-meshes or tensor-grids) of the PDE domain. We describe these strategies in terms of three distinct phases: the *partitioning*, the *allocation*, and the *message scheduling* . In the *partitioning phase*, we decompose the geometric data structures in a specified number (usually equal to the given number of processors) of subdomains or substructures so that :

(i) the subdomains have the "same" number of elements or grid points,

(ii) the number of interfaces among the subdomains is "small,"

(iii) the number of adjacent subdomains is "minimal,"

(iv) each subdomain is a connected domain.

In the *allocation phase* the objective is to allocate these subdomains to processors, so that :

(v) geometrically neighbor subdomains are allocated to neighbor processors in the interconnection network of a targeting parallel machine.

In the *message scheduling phase* the objective is to decouple (color) the processors so that :

(vi) the local synchronization among the processors introduces minimum edge contention in the network.

First, we present an algorithmic and software infrastructure consisting of "fast" heuristics for determining optimal geometry based mappings of PDE data suitable for matrix and domain decomposition methods. Furthermore, we describe a software system which assists the user in visualizing and manipulating such mappings in the Parallel-ELLPACK environment.

Second, we present a mapping methodology which consists of a set of well defined linear algebra primitives formulated on the PDE algebraic data structures and implemented on various targeting parallel architectures. In this methodology, known as BLAS (Basic Linear Algebra Subroutines), the solvers are implemented using these parallel BLAS primitives. In this thesis we have considered the parallelization of matrix-vector and matrix-matrix operations for banded matrices on distributed memory multiprocessor systems that support mesh and ring interconnection topologies. For their implementation we have employed systolic type techniques to eliminate synchronization delay and minimize the communication overhead among processors.

Furthermore, we analyze the theoretical complexity of our algorithms for the parallel BLAS and present some performance data on the nCUBE-6400 with 64 processors.

1. INTRODUCTION

Partial Differential Equations (PDEs) are the fundamental mathematical tools for describing the physical behavior of many applications in science and engineering. Most of the existing PDE software systems deal primarily with the solution of specific classes of PDE problems on sequential or vector machines. In this thesis we consider the study and implementation of two general parallel methodologies for solving PDEs on distributed and share memory MIMD machines. The techniques and software tools developed and analyzed in this thesis have been applied to general second order elliptic PDEs defined on 1, 2 and 3 dimensional domains. They can easily be extended to computations associated with the numerical simulation of more complicated "steady-state" mathematical models. The structure of the PDE problem assumed throughout this thesis is depicted in Figure 1.1.

1. Domain $\Omega \in R^n$, $n = 1, 2, 3$
2. Operator Equation : $Lu = f, x \in \Omega$
3. Auxiliary Conditions : $Bu = g, x \in \vartheta\Omega$

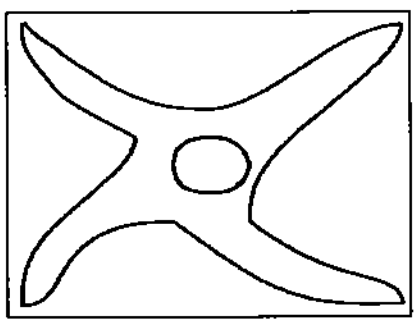


Figure 1.1 The components of a typical continuous PDE problem and an example of continuous domain Ω in R^2.

The first methodology considered here is based on the decomposition of the continuous or discrete PDE domain of definition in non overlapping substructures or subdomains (see [GGMP88], [CSS86], [CR87] and [KG87]). In the case of the splitting of the continuous domain, the original PDE problem is reduced to a set of "smaller" PDE problems defined on each subdomain whose auxiliary conditions have been "artificially" extended on the interior subdomain interfaces. The components of the decomposed PDE problem are depicted in Figure 1.2. Continuity requirements between subdomains are handled by an iterative technique over the subdomains. The proof of the equivalence of the decomposed PDE problem to the original one is not trivial. It depends very much on the artificial conditions employed and the operator L. The theoretical results in this area are limited.

1. Substructure : $\{\Omega_i\}_{i=1}^{P}$
2. Operator Equation : $Lu^i \mid_{\Omega_i} = f \mid_{\Omega_i}, i = 1, ..., P$
3. Boundary Conditions : $Bu^i \mid_{\vartheta\Omega_i} = f \mid_{\vartheta\Omega_i}, i = 1, ..., P$

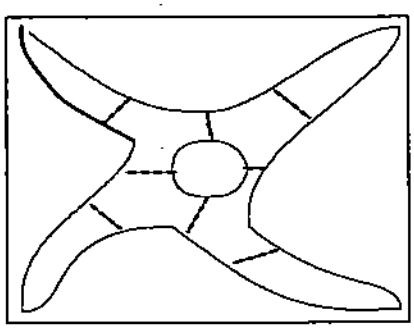


Figure 1.2 The components of the decomposed PDE problem based on the splitting of the domain Ω and an example of a substructure of the domain Ω.

In the case of the discrete PDE problem, this technique is applied on the splitting of the mesh or grid of the PDE domain which results into a splitting of the corresponding algebraic data structures consisting of the discrete equations corresponding

to the nodal or grid points of the subdomain and its interface (boundary). Figure 1.3 describes the decomposition of the discrete PDE problem.

1. Submeshes : $\{\Omega_i^h\}_{i=1}^P$
2. Subsystem of Equations : $K_1^i x^{\Omega_i^h} = f_1^{\Omega_i^h}, i = 1, ..., P$
3. Interface Equations : $K_2^i x^{\vartheta\Omega_i} = f_2^{\vartheta\Omega_i^h}, i = 1, ..., P$

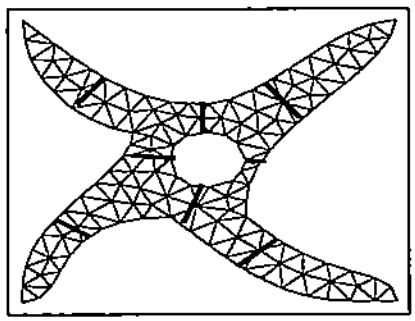


Figure 1.3 The components of the decomposed discrete PDE problem based on the splitting of the mesh or grid used in the numerical simulation and an example of a decomposition of a finite element mesh Ω^h.

Throughout this thesis, we refer to the first approach as the *continuous domain decomposition* approach and the second one as the *discrete domain decomposition* approach for solving PDEs. The structure of this general solution framework of PDEs is inherited parallel and suitable for MIMD machines that support course grain parallelism. Figure 1.3 suggests a parallel formulation of this methodology which we have implemented in the parallel ELLPACK system [HRC+90] and nCUBE-6400 machine. Notice that the mapping of the underlying computation is based on the decomposition of the geometric data and their optimal mapping to the targeting architecture. In this formulation the splitting of a continuous/discrete domain and its mapping to the targeting architecture is viewed as a set of parameters to be determined by the user. Unfortunately, the performance of the underlying computation depends very

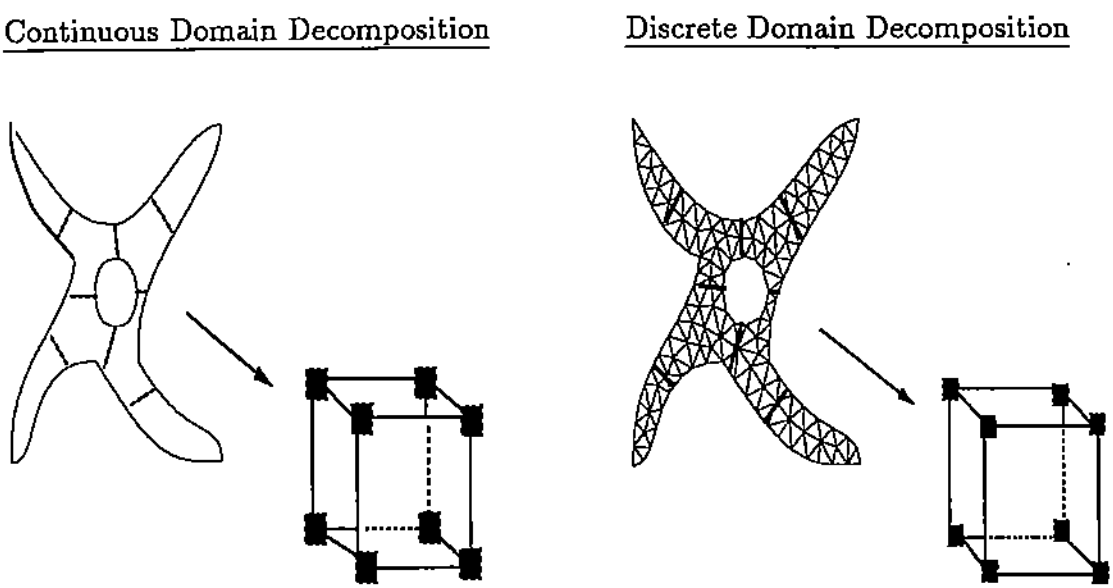


Figure 1.4 A parallel MIMD PDE solution methodology based on the domain decomposition approach.

much on the optimal selection of this set of parameters, which turns out to be an NP-Complete problem.

The first objective of this thesis is to develop a software environment (i.e., a set of data structures and module interfaces) that allows the implementation of the domain decomposition methodology and the second objective is to develop "fast" algorithms to determine these parameters and tools to visualize and manipulate them. The results of our investigation in this area are reported in Chapters 2, 3, and 6.

The second parallel methodology we explore in this thesis is the one that is based on a virtual language consisting of primitive algebraic operations known as BLAS, which can be used to implement numerical PDE solvers. These primitives can be implemented on different parallel machines, thus achieving portability of the code across many targeting architectures. In this thesis, we consider the parallel implementation of the so called level 2 and 3 BLAS on the nCUBE-6400 for dense and sparse data

structures and we report on their performance. The results of this investigation are reported in Chapter 5.

2. GEOMETRY BASED MAPPING TECHNIQUES

In this chapter we develop and analyze strategies for *mapping* discrete PDE domain decomposition solvers onto MIMD machines so that (a) the workloads of all processors are balanced and (b) the processor synchronization and communication costs are kept to a minimum. In the case of PDE computations, such mappings can be formulated at three distinct levels: the discrete geometrical data structures associated with the PDE domain (mesh or grid), the discrete algebraic equations associated with some discretization of the PDE equations (sparse system of algebraic equations), and the data flow graph of the PDE solver. We primarily study mapping techniques formulated at the geometric data structures of the PDE problem. The mapping methodology employed is viewed in terms of three distinct phases corresponding to the : *partitioning and allocation* of the discrete PDE geometric data and the *scheduling* of the communicating data in the underlying computation.

In the *partitioning phase* we decompose the geometric data structures to a prespecified number (usually equal to the number of processors) of subdomains or substructures such that the following criteria are approximately satisfied:

(i) the subdomains have the same number of elements (finite element meshes) or grid points (grids),

(ii) the number of interface points (i.e., the number of node points at the boundaries between subdomains) is small relative to the total number of subdomain node points,

(iii) the number of adjacent subdomains is minimum, and

(iv) each subdomain is a connected domain.

In the *allocation phase* the objective is to assign these subdomains to processors, such that the following objective is satisfied:

(v) the communication requirements of the underlying computation between the processors of a given architecture are minimum.

Finally, in the *scheduling phase* the objective is to schedule the local messages such that the following objective is satisfied :

(vi) the edge-contention of the network is minimum.

In section 2.1 we present a mathematical formulation of the partitioning, allocation and scheduling phases. In section 2.2 we review the partitioning heuristics that presented in the literature the last fifteen years and we present new partitioning clustering and optimization based heuristics. In section 2.3 we devise new allocation strategies and present a number of performance criteria under which these strategies are evaluated. In section 2.4 we present a scheduling heuristic. The results of this study have been already published in [CHENHR89], [HRC+90], [CHH91], [CHENH+91], and [CR92].

2.1 Formulation of the partitioning, allocation and scheduling phases.

As mentioned above, we want to minimize the synchronization and communication costs of the parallel PDE iterative solvers [CHK+92] based on discrete domain decomposition methods by finding an optimal solution for the partitioning, allocation and scheduling of the computation. To be able to find such a solution first, we analyze the parallel computation of these solvers and then, we model the partitioning, allocation, and scheduling phases.

(a) Assumptions and Definitions

Below we state our assumptions and definitions related to the mapping of geometric data structures on distributed memory MIMD circuit switching machines, with a

fixed routing mechanism. First, we assume that the targeting parallel machine consists of a network of processors connected by communication links and we denote this interconnection network by $G_A(V_A, E_A)$, where the vertices (V_A) are the processors and the edges (E_A) are the communication links between the processors. Each processor exchanges information in groups of bits called *packets* using the communication links of the network. The length of the packets varies form few tens of bits [nCU91] to several thousands of bits [iPS90]. The bits of a packet are consecutively transmitted without interruption. The process of sending or receiving a message which is stored in a buffer can be viewed as a transmission of a number of packets. The local memory of each processor is used for storing some problem data and intermediate results (in its local data structures).

In addition, we define as geometric data a finite element mesh Ω_h consisting of: a set of elements $\{e_j\}_{j=1}^{NE}$ with nodes $\{n_i\}_{i=1}^{N}$, where NE is the number of elements in the mesh and N is the number of nodal points in the mesh. For each node n_i we define the connectivity ω_i of the node as the number of the adjacent nodes. The mapping of finite difference grids can be done following analogous steps. Some of the mapping techniques we present later can be formulated and implemented on the so called mesh graph denoted as $G_M(V_M, E_M)$ where the vertices (V_M) correspond to the elements or nodes of the mesh and the edges (E_M) indicate the connectivity of the element or node with its neighbors. It is worth noticing that the nodal mesh graph is identical to the mesh.

(b) Communication requirements of the parallel PDE iterative solvers

The iterative PDE solvers for the solution of a discrete linear system of algebraic equations can be reduced into matrix-vector multiplication operations (see [HY81] and [KRYG82]). The parallel processing and implementation of matrix-vector multiplication operations consists of two steps : (a) the *local communication* and (b) the *local computation* (see [FJL88], [CAHH92]). Thus, the parallel PDE synchronous iterative solvers based on the discrete domain decomposition methods require in each iteration

a *local communication* that is necessary for the synchronization of the iterative solver. Throughout this thesis, we also refer to it as *local synchronization.*

A high level view of the steps of an iterative solver that preserves the ordering of the corresponding sequential computation, for the *discrete domain decomposition* methods pertinent to the mapping issue is the following :

(i) *Local Synchronization*,

(ii) *Local Computation*, and

(iii) *Global Synchronization.*

In this work we address only the local synchronization issue, as local computation and global synchronization issues have been fully investigated by many established researchers in the area. The local synchronization consists of an exchange of messages between the processors of the parallel machine; the messages transfer some of the local data (i.e., interface unknowns) required by the geometrical neighbor subdomains. Figure 2.1 illustrates the interface nodes associated with the interface unknowns of the discrete domain decomposition method used for the parallel numerical solution of a Poisson problem defined on the domain Ω which is discretized by a bilinear finite element method. The local computation mainly consist of matrix-vector and vector-vector operations. Finally, the global synchronization method consist of global communication operations that are required for the acceleration of the convergence and for the checking of stopping criteria [CHK+92].

The local synchronization mechanism used for the parallel processing of the iterative solvers based either on discrete or continuous domain decomposition methods is described by the algorithmic segment of Figure 2.2.

The execution time of the local synchronization segment depends on a number of factors. In this paragraph we analyze some of the factors that contribute in the time complexity of the local synchronization scheme of Figure 2.2 for parallel systems. The execution time of the above local synchronization scheme for the processor

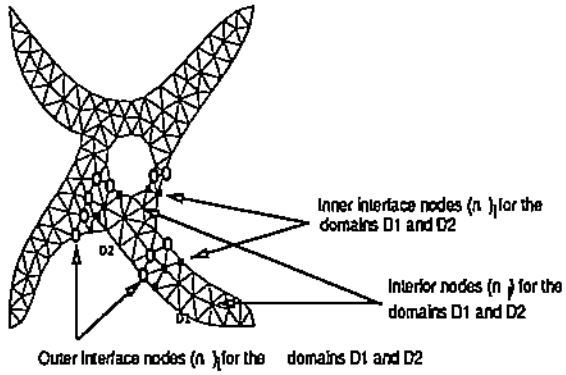


Figure 2.1 Interface (inner and outer) and interior node points for a discrete domain decomposition of the domain Ω.

Local Synchronization Segment (LSS) :

(i) Copy inner interface unknowns from local data structures to a buffer (T_{copy}).

(ii) For each $D_j \in N(D_i)$ send $S_{buffer(j)}$ to the processor $m(D_j)$ (T_{send}).

(iii) For each $D_j \in N(D_i)$ receive the $R_{buffer(j)}$ from the processor $m(D_j)$ (T_{recv}).

(iv) Copy the outer interfaces $R_{buffer(j)}$ $\forall D_j \in N(D_i)$ to local data structures. (T_{copy}).

Figure 2.2 Local synchronization (or communication) segment for the domain decomposition based iterative PDE solvers.

$m(D_i)$, is decomposed into three components, namely the time to send (T_{send}) a set of messages to processors $\Pi(m(D_i)) = \{m(D_j)\ \ processor,\ \ where\ D_j \in N(D_i)\}$, the time to copy ($T_{copy}$) the local data structures into and from a buffer, and the time to receive (T_{recv}) the messages from the set of processors $\Pi(m(D_i))$. Thus, the total execution time of the local synchronization segment for the implementation of the domain decomposition based iterative PDE solvers can be model by :

$$T_{LS}^{m(D_i)} = 2T_{copy} + T_{send} + T_{recv} \tag{2.0}$$

In this relation the T_{send} is the time required by the processor to assemble the message and move it to the appropriate buffer. Assembling information for the message includes tasks like appending and addressing information as well as selecting a link on which to transmit the message. T_{send} depends on architectural parameters like packet or circuit switching mechanisms, size of the message buffer, and resource management (i.e., queing time $\sum_{D_j \in N(D_i)} Q_{m(D_i),m(D_j)}$), as well as on problem parameters like the number of neighbor subdomains (i.e., $\mid N(D_i) \mid$) and length of interface points (i.e., $\sum_{D_j \in N(D_i)} c(D_i, D_j)$). The T_{copy} mainly depends on the size of the interface length (i.e., $\sum_{D_j \in N(D_i)} c(D_i, D_j)$). Finally, the time T_{recv} depends on the difference between the actual and expected times of message arrivals; note that for almost all commercially available distributed memory MIMD parallel machines the receive operation is a blocking operation.

(c) Mathematical formulation of the partitioning phase

The partitioning of Ω_h into P non-overlapping subdomains $\{D_i\}_{i=1}^{P}$ is characterized in terms of the set of geometric neighbor subdomains $N(D_i)$ to subdomain D_i and the number of interface nodes (interface length) $c(D_i, D_j)$ between the subdomains $D_i\ \ and\ \ D_j$. By considering the subdomains as nodes (super-nodes) of a mesh, we can overload the notation $G_M(V_M, E_M)$ where the vertices (V_M) correspond to super-nodes (subdomains) of the mesh and the edges (E_M) indicate the connectivity of the

super-nodes with its neighbors. Then, the optimal partitioning, as defined by criteria (i) to (iii), can be viewed as the one with :

$$\min \max_{D_i \in V_M} deg(D_i) \quad and$$

$$\min \max_{D_i \in V_M} \sum_{D_j \in N(D_i)} c(D_i, D_j)$$

whose subdomain size $|D_i|$ satisfies the following constraint :

$$\lfloor N/P \rfloor \leq |D_k| \leq \lceil N/P \rceil \quad k = 1, \ldots, P \tag{2.1}$$

where $|D_k|$ is the size of the subdomain D_k and it is defined as the cardinality of the set of mesh nodal points that belong in D_k. We can easily show [CHENHR89] that the determination of a partitioning that satisfies the criteria (i) and (ii) can be reduced to solving the following constraint optimization (minimization) problem :

$$\min \frac{1}{2} \sum_{k,\ell=1}^{P} \sum_{e_i \in D_k} \sum_{e_j \in D_\ell} \chi(e_i, e_j) \tag{2.2}$$

subject to the constraint (2.1), where

$\chi(e_i, e_j) = 1$ if e_i and e_j are adjacent and in different subdomains

$= 0$ otherwise.

The criteria (iii) and (iv) are imposed implicitly during the determination of the solution of (2.1) and (2.2) [CHENHR89] by seeking solutions that optimize certain additional functions known as *profit* functions.

(d) Mathematical formulation of the allocation phase

The determination of an optimal allocation m is equivalent to minimizing the communication overhead. The overhead due to communication of non-local data can be modeled in terms of the length of the interfaces between the subdomains and the distance of the processors to which the subdomains have been allocated. The mathematical model for the allocation phase is described by the following two minimization expressions :

$$\min_m \frac{1}{2} \sum_{i=1}^{P} \sum_{j=1}^{P} c(D_i, D_j) d(m(D_i), m(D_j)) \tag{2.3}$$

or

$$\min_{m} \max_{1 \le i \le P} \{ \sum_{D_j \in N(D_i)} c(D_i, D_j) \times d(m(D_i), m(D_j)) \} \tag{2.4}$$

where $d(m(D_k), m(D_\ell))$ is the distance between the two processors $m(D_k), m(D_\ell)$ assigned to D_k and D_ℓ in the interconnection network (graph G_A) of the parallel MIMD machine. The distance between two processors $m(D_k)$ and $m(D_\ell)$ in the interconnection graph of a distributed memory parallel machine is defined to be equal to the length of the path I_0, where

$I_0 = \min_{length\ of\ I} \{I,$ *path that connects the nodes* $m(D_k)$ *and* $m(D_\ell)$ *in the graph* $G_A\}$.

By definition, the length of a path between two nodes is the number of the edges in the path. In the case of a hypercube or mesh graphs, the distance between two nodes $m(D_k)$ and $m(D_\ell)$ is equal to the Hamming distance $\mathrm{H}(m(D_k), m(D_\ell)) = \mathrm{i}$, where i is the number of different bits in their binary representation [SS88].

(e) The formulation of the scheduling phase

The final mapping phase is attempting to compute a scheduling of the local messages that are interchanged among the processors. A well known problem for circuit switching networks with fixed routing scheme is the edge-contention problem which results when messages share common communication links. The scheduling of the local messages minimizes the edge-contention of the network. The formulation of this phase is given in terms of the requirements of the communication protocol, described by the local synchronization segment of Figure 2.2, for the iterative solvers based on the domain decomposition approach.

Although the six criteria can be imposed independently of each other, for some applications, it is advisable to combine them with appropriate weights [Fox86], [FOS88], [Wil90], [Man92].

2.2 Partitioning heuristics

In this section we present several strategies for solving the optimization problem defined by (2.1) and (2.2). It has been observed that this optimization problem is an NP-Complete problem [Gare 79]. Thus, in this thesis we present various "fast" heuristics for the partitioning of finite element and difference meshes.

2.2.1 Clustering strategies

In this section we outline the basic idea of clustering methods and we present an overview for some of the clustering techniques that can be used for the partitioning of PDE computations. The objective of clustering methods in general is the discovering of a structure within complex bodies of data. In a typical example one has a sample of data units like persons, nodal points, and finite elements. Each described by selected attributes like human-characteristics, coordinates, and connectivity. The goal is to group the data units into clusters such that the data units within a cluster have a high degree of "natural association" among themselves while the clusters are "relatively distinct" from each other. The approach to the problem and the results achieved depend principally on how the investigator chooses to give operational meaning to the phrases "natural association" and "relatively distinct."

(a) Clustering strategies based on rooted level structures

One such class of clustering methods is the class of *reordering* methods that have been developed to preserve sparsity in Gaussian elimination for symmetric sparse matrices. One of the major challenges for researchers in computational sciences is the development of efficient storage schemes and fast solvers for a system of n linear equations in n unknowns denoted as :

$$Ax = b,$$

where A is an n by n matrix of constants, x is the vector of unknowns and b is a vector of constants. If A is symmetric and positive definite, then Gaussian elimination can be used to factor the matrix A into a product of the form LDL^T, where L is a lower

triangular matrix with ones on the diagonal and D is a diagonal matrix. Then it is easy to solve for x by solving the two triangular systems $Ly = b$ and $DL^T x = y$. The complexity of using Gaussian elimination to factor the matrix A and find x depends on the sparsity of the matrices A and L. This scheme is successful in the case that L is sparse. The problem is reduced in finding an elimination order (i.e., a permutation of the rows and columns of the matrix A that gives the smallest possible fill-in. It has been shown that this is an NP-Complete problem [Gil80]. In [LS76], [Geo73], [GL78], and [GM78] several *reordering* heuristics like *Cuthill McKee, reverse Cuthill McKee, automatic nested dissection, and minimum degree* have been introduced for its solution. Generalizations of these algorithms appear in [Gil80] and [Liu89b].

The Cuthill McKee ordering scheme and the automatic nested dissection are instances of a more general class of a partitioning scheme based on a structure known as *rooted level structure*. For a given connected mesh graph G(V,E) and a nodal point x in V, the rooted level structure at x is defined to be the sequence of nodal subsets :

$$L_0, L_1, L_2, ..., L_n$$

where $L_0 = \{ x \}$, and $L_j = Adj_G(\cup_{k=0}^{j-1} L_k)$, with
$Adj_G(\cup_{k=0}^{j-1} L_k) = \{v \in V - \cup_{k=0}^{j-1} L_k$ that are adjacent to a vertex $y \in \cup_{k=0}^{j-1} L_k \}$
for level $j = 1, ..., n$. The value n is the length of the longest path from the node x to other nodes in the graph. The node $x \in V$ with the longer rooted level structure, i.e., node with largest possible eccentricity, is called *peripheral* node. No linear algorithm is known for finding a peripheral node. George and Liu in [GL81] presented a heuristic for finding nodes of high eccentricity (pseudo-peripheral nodes). The following algorithm outlines a 2-way partitioning scheme based on the rooted level structure.

Algorithm 2.2

begin

1. Compute a peripheral node.
2. Partition the nodes into levels $L_0, L_1, L_2, ..., L_n$.
3. Separate the vertices V in the level $s = \lfloor (n+1)/2 \rfloor$
4. Choose a minimal subset of L_s that is still a separator.

end

In [Liu89a] Liu presented a graph partitioning algorithm that finds an initial separator based on minimum degree ordering and then it iteratively improves the initial partition by graph matching. Liu compared his new algorithm and the rooted level structure partitioning algorithm and he observed consistently better partitions. Both Liu's algorithm and the rooted level structure partitioning algorithm can be used to partition a mesh into P connected submeshes (subdomains).

An attempt to generalize the above elimination *reordering* techniques to solve the P-way partitioning problem for connected graphs made by Farhat in [Far88]. Farhat presented an algorithm that is a generalization of the 2-way rooted level structure partitioning algorithm for P-way partitions. The algorithm produces load balanced partitionings with minimum amount of interface points among the subdomains and handles domains with irregular geometry and arbitrary discretization, but it does not avoid partitionings with subdomains consisting of more than one simply-connected components (see Figure 2.3). This algorithm will be referred as CM_Clust throughout this thesis and it is outlined below.

Algorithm 2.3

begin

for k = 1 *until* P do

1. Locate a node n_i on interior boundary of D_{k-1} with $\min_i w_i > 0$.
2. Initialize D_k with all unassigned elements connected to n_i.
3. Recursively, for each element $e_j \in D_k$ do
   - reduce the current weight of each node attached to e_j,
   - assign to D_k all unassigned elements adjacent to e_j.
4. Repeat steps (1) to (3) until $|D_k| < c_k$.
5. Mark all nodes belonging to the interior boundary of D_k.

endfor

end

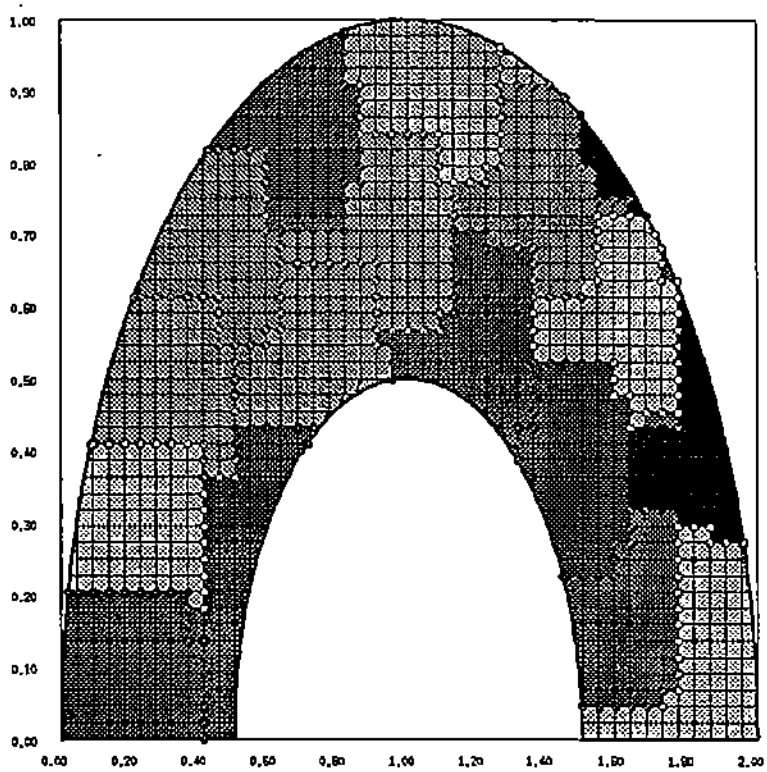


Figure 2.3 16-way partitioning of a discretized semi-annulus domain using the algorithm proposed by Farhat in [Far88].

In [ANN90] Al-Nasra et al. have attempted to improve Farhat's algorithm by using both the topology and the geometry of the mesh to avoid the splitting of the subdomains. They modify the first step of the for loop, of the Algorithm 2.3, by introducing an additional weight for the nodes of the mesh. They calculate the long and short directions of the two dimensional domain and they adjust the nodal weight by using the following formula :

$$\omega_i := \omega_i + p^2 * (\frac{\delta}{a}) * (\frac{a}{b} - 1)$$

where ω_i is the node connectivity, δ is the step size of the mesh along the long direction of the smallest rectangular, say R, that encloses the domain, and a, b are the sizes of R with $a := \max\{a, b\}$. They claim that the undesirable splitting of the subdomains did not occur for all the problems they tested. Figure 2.4 shows the partitioning of a simple (L-shape) 2D domain enclosed by a square (i.e., $\frac{a}{b} - 1 = 0$).

Both algorithms described above are of linear time complexity, satisfy the criteria (i) and (ii) for non-convex domains, fail to generate connected subdomains with relatively small connectivity, and are sensitive to a prior enumeration of the nodes (or elements) and the starting node (or element). Figure 2.5 illustrates the partition of the L-shape domain for two different enumerations of the nodal points.

(b) Clustering strategies based on strip or block partitioning

Another simple and attractive clustering method considered by many researchers (see [SE87a], [FOS88], [LF90] and [PAF90]) is the so-called strip or block partitioning heuristic. This heuristic is referred under different names, some of them are : one-dimensional (1D) strip partitioning, two-dimensional (2D) strip partitioning, multilevel load balanced method, median splitting, and sector splitting. Throughout this thesis, we are referring to this clustering algorithm with the following two names : (i) RxQ, where R is the number of subdomains (blocks or strips) along the x-axis, Q is the number of subdomains (blocks or strips) along the y-axis, and R $\times$ Q = P (for 2D domains) and (ii) RxQxS, where R is the number of subdomains (blocks or strips) along the x-axis, Q is the number of subdomains (blocks or strips) along the y-axis,

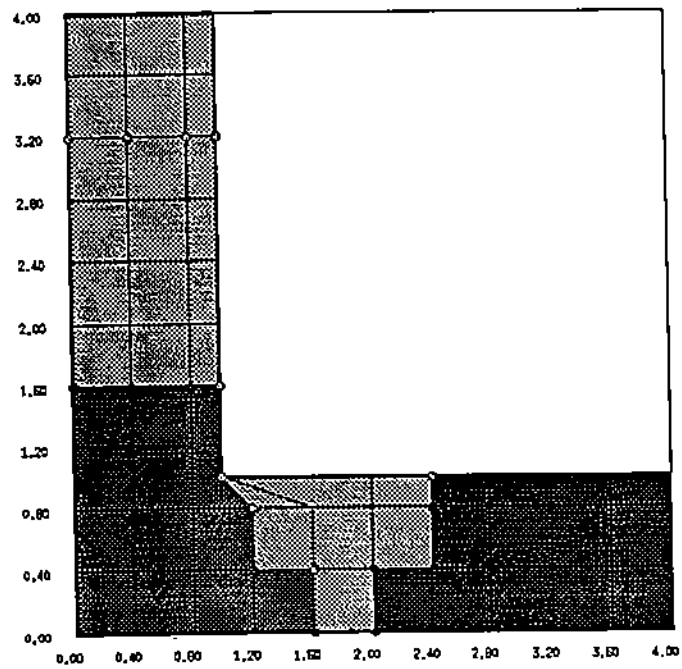


Figure 2.4 4-way partitioning of a discretized L-shape domain obtained by the algorithm introduced in [AlNT90].

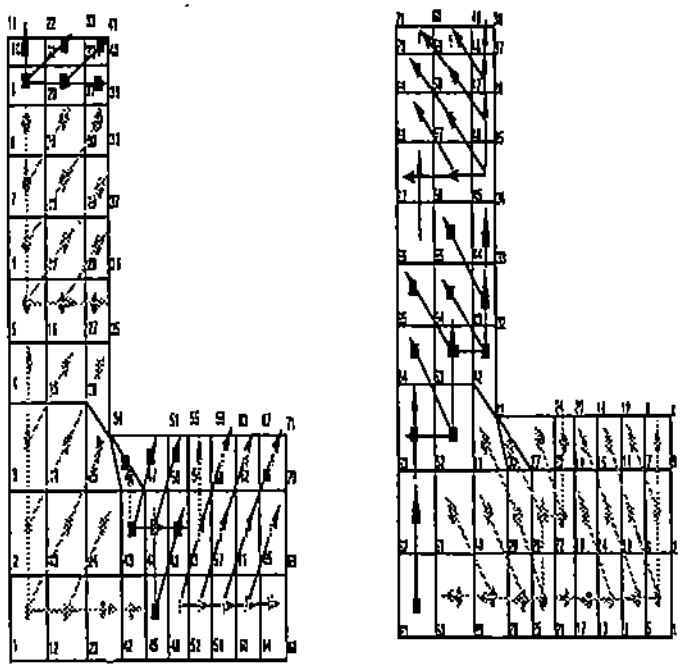


Figure 2.5 Two different 4-way partitionings of the same domain based on two different enumerations of the nodal points of a quadrilateral mesh.

S is the number of subdomains (blocks or strips) along the z-axis, and R × Q × S = P (for 3D domains). The RxQ heuristic partitions the nodal points of the mesh into R strips (subdomains) along the x-axis and then it partitions each subdomain into Q strips along the y-axis. The RxQ heuristic many times partitions non-convex 2D domains into subdomains with more than one simply-connected components. Figures 2.3, 2.6 and 2.7 illustrate the partitioning of orthogonal and triangular meshes of the semi-annulus 2D non-convex domain.

The advantages of the RxQ heuristic are : it is of O(nlogn) time complexity, it satisfies criteria (i) and (ii), it is not sensitive to a predefined enumeration of the nodes (or elements), and it is suitable for the mapping of the subdomains onto linear array and 2D-mesh architectures. A disadvantage of the RxQ heuristic is that in some cases it fails to generate connected subdomains for non-convex domains (see Figure 2.6).

(c) Clustering strategies based on boundary-conforming curvilinear coordinate systems

Most of the existing partitioning heuristics fail to partition non-convex domains with complex boundary shapes into connected subdomains with small connectivity. In this Section we present clustering heuristics based on attributes that characterize the boundary shape (geometry) of the physical domain. We generalize the R×Q partitioning heuristic by using attributes associated with the curvilinear coordinate system that is defined by a boundary-value problem on the physical domain. This idea is based on numerical mesh generation and provides the key to remove the problem of boundary shape from finite difference and element methods. The numerical mesh generation algorithms are using mathematics (PDEs) to control the placement of the mesh points so that the functions based on them can represent the physical solution [WM85].

The partitioning of a relatively more complex domain, such as the one in Figure 2.8, is based on the clustering of the nodes (or elements) first along the x^*-axis and then along the y^*-axis of the Cartesian coordinate system (x^*, y^*), where (x^*, y^*) are

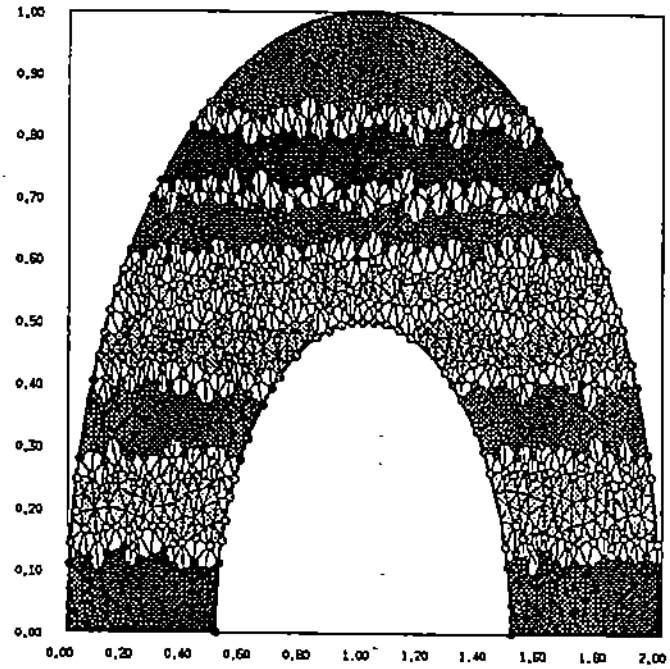


Figure 2.6 8-way partitioning of a discretized semi-annulus domain using the RxQ heuristic, with R = 1, and Q = 8.

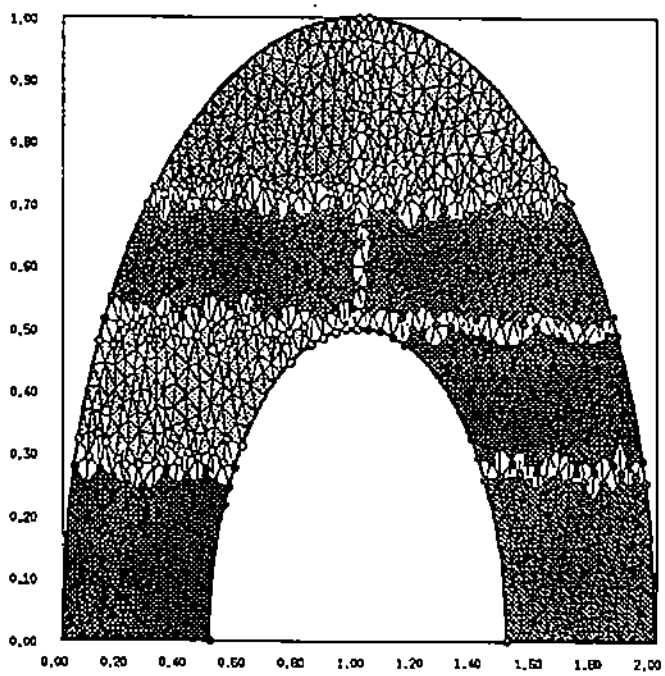


Figure 2.7 8-way partition of a discretized semi-annulus domain using the RxQ heuristic, with R = 2, and Q = 4.

computed by the following transformation :

$$\begin{bmatrix} x^* \\ y^* \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

By applying the same clustering idea using cylindrical coordinates (r, θ), where

$$r(x,y) = \sqrt{(x^2+y^2)}, \qquad \theta(x,y) = tan^{-1}\frac{y}{x}$$

we get the partition of Figure 2.9.

We can use *boundary-conforming curvilinear coordinate systems* to generalize the above clustering heuristics for more general 2D (or 3D) simply-connected domains. The way to accomplish this for P = R×Q processors is : (1) sort the nodal points (or elements) along the coordinate lines conforming to the boundaries (analogous to the way in which lines of constant radial coordinate coincide with circles in cylindrical coordinate system) and (2) group the nodal points (or elements) into R subgroups and then, sort the points of each subgroup along the other curvilinear coordinate (analogous to the angular coordinate in the cylindrical coordinate system). This coordinate varies monotonically along the boundary. Finally, group the nodal points (or elements) of each of the R subgroups into Q subgroups. Figure 2.10 illustrates the curvilinear lines of a 2D curvilinear coordinate system and shows a 16-way partitioning based on these curves.

(d) Clustering strategies based on scattered decomposition

An even simpler clustering approach, called *scattered decomposition*, is presented in [MO87]. The *scattered decomposition* is a generalization of the RxQ partitioning for irregular 2D domains. This heuristic consists of the following two steps : (i) the embedding of the interconnection graph (in [MO87] they considered the hypercube interconnection) to a two-dimensional processor lattice and (ii) the covering of the mesh with several copies of this processor lattice. Many disconnected submeshes are assigned to a single processor. Before we investigate the performance of the scattered decomposition, we adopt the terminology introduced in [MO87]. A single copy of the

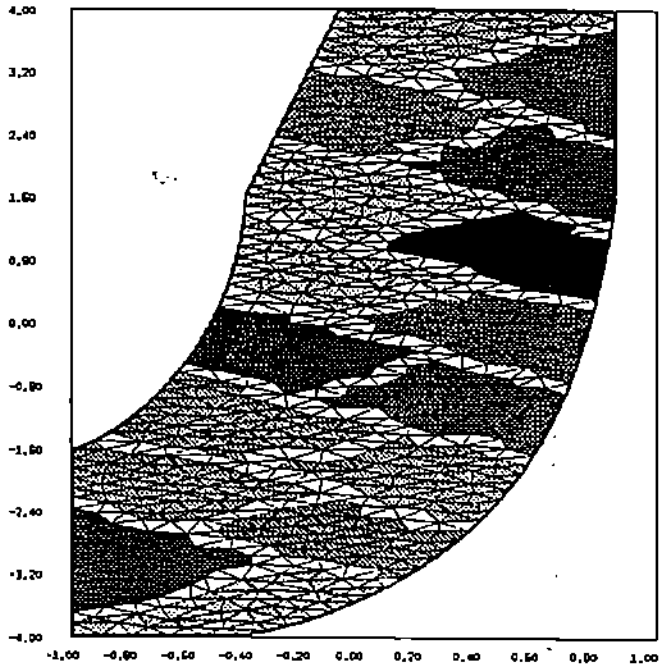


Figure 2.8 16-way partition based on attributes associated to (x^*, y^*) coordinate system.

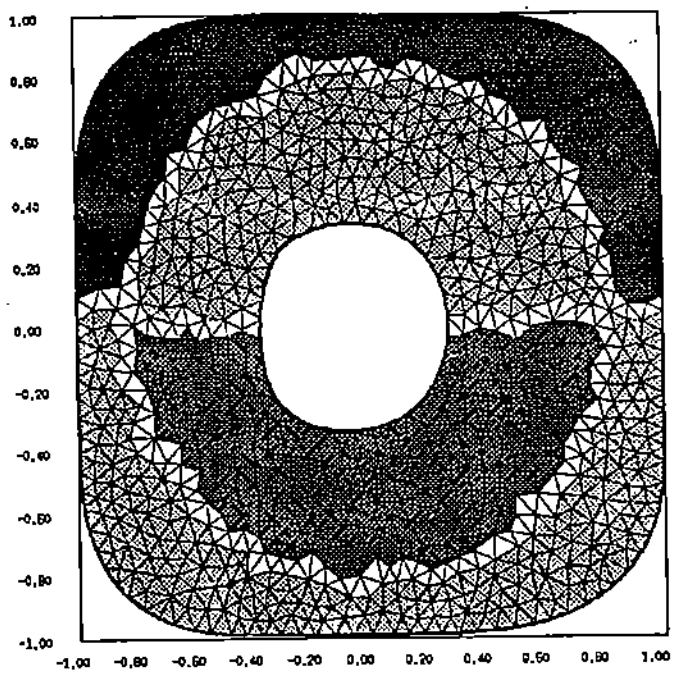


Figure 2.9 4-way partition based on the cylindrical coordinates.

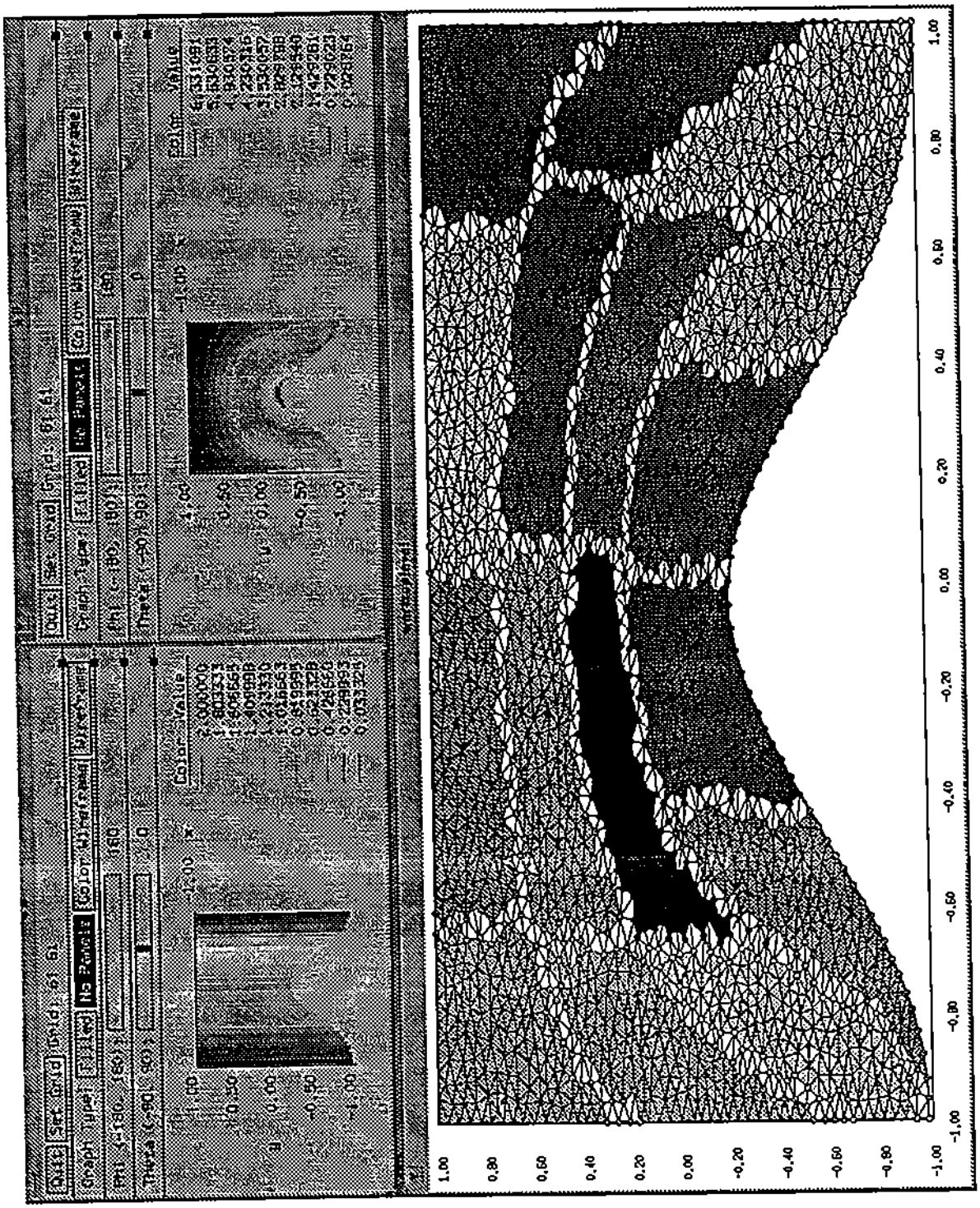


Figure 2.10 left) A 2D curvilinear coordinate system (ξ, θ), and right) 16-way partition based on this coordinate systems (ξ, θ).

fundamental processor lattice is called a *template* and a processor's assignment within a single template is called a *patch*. Figure 2.12 shows the mesh of Figure 2.11 with scattered decomposition.

The advantage of the scattered decomposition is the ability to map a large class of irregular scientific computations (see in [CT88]) without ever analyzing them. Although scattered decomposition is an inexpensive way to load balance irregular computations (by using patches of fine granularity), its main disadvantage is the higher communication cost due to fine granularity of the mapping (see in [CT88]).

2.2.2 Deterministic optimization strategies

The oldest and most general approach for the solution of difficult combinatorial optimization problems is the *local (or neighborhood) search*. The idea is simple and it is successfully applied to a variety of difficult combinatorial optimization problems [PS82].

In a typical combinatorial optimization (CO) problem each instance of the CO problem is associated with a finite set of feasible solutions. Each feasible solution is associated with a cost which is the value of the objective function to be optimized at the solution point. The goal is to find a solution that minimizes or maximizes the cost. Local search algorithms for CO problems require the definition of a neighborhood structure for each solution i.e., a finite set of solutions which are in some sense "close" to that solution. For example, in the mesh partitioning problem of finite element mesh m, an obvious neighborhood of a given partition (m_1, m_2) of the mesh m is the finite set

$$\{(m_{1_i}, m_{2_i}), \text{ where } m_{1_i}, m_{2_i} \text{ are connected meshes}$$
$$\text{and } m_{1_i} = (m_1 - \{x\}) \cup \{y\} \text{ and } m_{2_i} = (m_2 - \{y\}) \cup \{x\} \text{ with } y \in m_2 \text{ and } x \in m_1\}.$$

Bellow we describe in detail the procedure followed by a local search algorithm for a given instance (f, c) of a combinatorial optimization problem, where f is the set

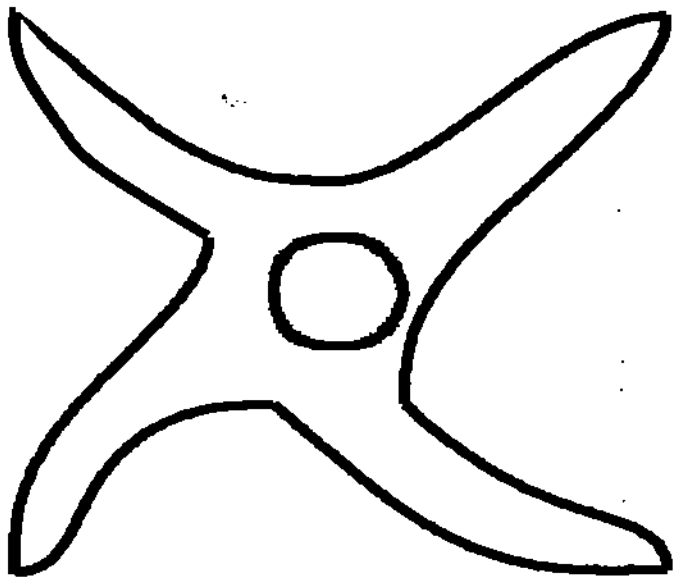


Figure 2.11 Two-dimensional irregular non-convex, connected domain with a hole.

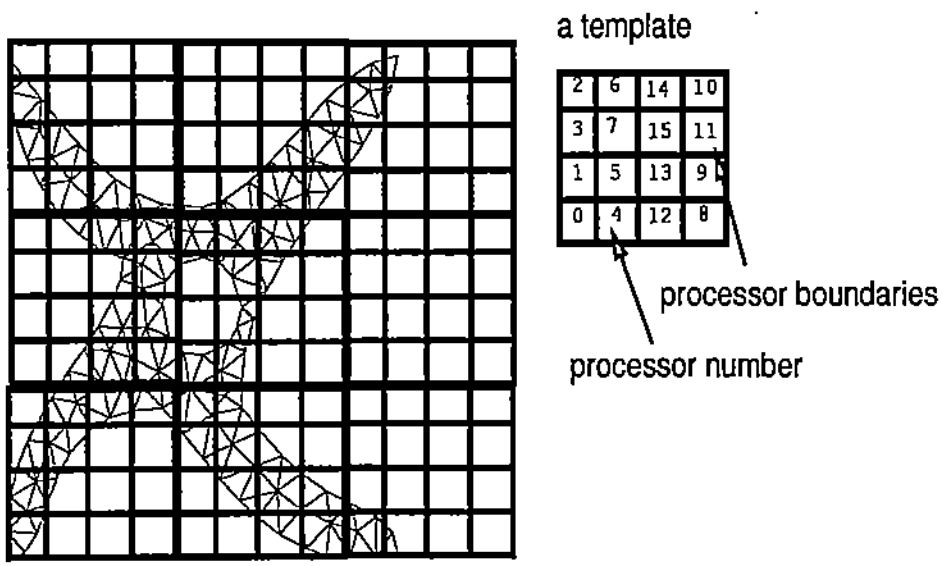


Figure 2.12 The scattered decomposition applied to a mesh of Figure 2.11. It uses a nine-template covering of the problem by a 16 processor machine.

of feasible solutions and

$$c : f \rightarrow \Re \tag{2.5}$$

is the objective function, where $\Re$ is the set of real numbers. First, we superimpose a neighborhood structure $N : f \rightarrow 2^f$ which is completely searched by the following function :

Algorithm 2.4

improve(t) = any u where u $\in N$ with cost(u) $\leq$ cost(t) if such an u exists
= null otherwise.

Second, we start at some initial feasible solution $t \in f$ and use Algorithm 2.4 to do a local search and repeatedly replace the current solution by a neighboring solution of a better value, until no such neighboring solution exists. At this point we have identified a solution that is "locally optimal." The above local search technique is illustrated by the following algorithm :

Algorithm 2.5

```
begin
   t := some initial starting point in f;
   while improve(t) ≠ null do
      t := improve(t);
      return t;
   endwhile
end
```

The only difference between various local search algorithms is in the definition of their neighborhood structures. Neighborhood structures can be defined either by complex relations among the feasible solutions or by a set of randomly chosen uniformly distributed points in the set of feasible solutions. Since the problem of partitioning the nodes of a mesh or grid is the same with the partitioning problem of a general graph, the neighborhood structures that have been defined for the graph partitioning problem can be used for the partitioning of PDE computations based on the discrete geometry of the physical domain.

Next, we review some of the neighborhood structures for the graph partitioning problem that appear in the literature (see [KL70], [Got81], [PK89] and [Swa92]). The simplest neighborhood structure, for the partitioning of the graph G(V, E) and an initial 2-way partitioning (A, B), is given by the following equation :

$$N_s(A,B) = \{\text{all partitionings } A^*, B^* \text{ that can be obtained from the partitioning A, B by a single swap operation }\},$$

where the *swap* operation of forming A^*, B^* is defined by :

$$A^* = (A - \{a\}) \cup \{b\}, \text{ and } B^* = (B - \{b\}) \cup \{a\}$$

with $a \in A$ and $b \in B$.

Kernighan and Lin in [KL70] generalized the above idea in forming a neighborhood structure for the local search by replacing a single swap operation by a sequence of swaps. Figure 2.14 (a) illustrates a geometrical representation of the local search with neighborhood structures based on a single swap operation (ordinary local search) and and Figure 2.14 (b) illustrates the local search with neighborhood structures based on a sequence of swap operations (variable depth local search). At each step of the sequence in their algorithm, the authors choose a swap involving a pair of unswapped vertices that yields the best cost. As Figure 2.13 illustrates the first few swaps might worsen the initial partitioning but they will help the local search to climb out of some local minima. The algorithm stops at any point with positive or maximum gain.

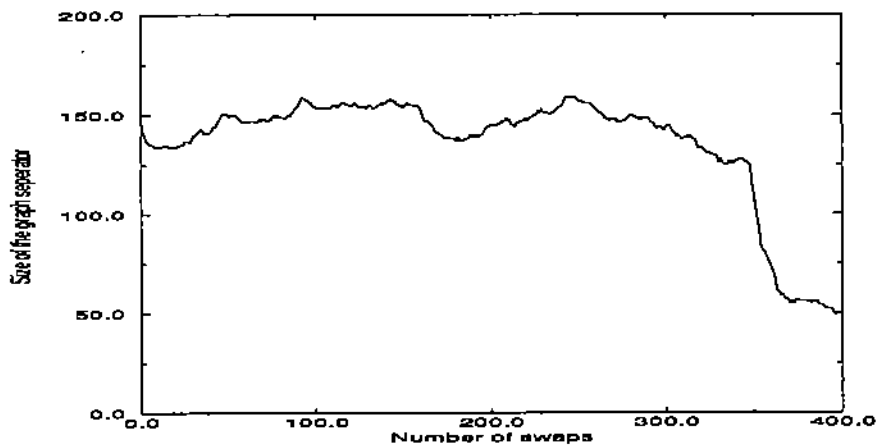


Figure 2.13 The behavior of the objective function (2.5) i.e., size of the separator, as a function of the number of swaps for the Kernighan and Lin local search algorithm.

Another more complicate generalization of the ordinary local search is presented by Satoshi Goto in [Got81]. Satoshi Goto replaced the pairwise swapping with an interchange of more than two vertices at the same time. We shall call this operation a *multi-swapping* operation. Goto presented the multi-swapping operation for the module placement problem in electrical circuit layout. The same extension can be used to define neighborhood structure for the P-way graph partitioning problem.

Finally, Lee et al. [PK89] and then Tao et al. [Swa92] presented a transformation of the bisection (and P-way) graph partitioning problem into the max-cut problem. Using this approach they were able to solve the partitioning problem by defining the primitive operation *move* :

$$A = A - \{a\}, \text{ and } B = B \cup \{a\}, \text{ where } a \in A$$

Two *move* operations are equivalent to one swap operation. As a result they were able to define a more general neighborhood structure for the 2-way (and P-way) partitioning problem :

$$N_{mm} = \{\text{all partitionings } A^*, B^* \text{ that can be obtained from the partitionings A, B by a sequence of } move \text{ operations }\}$$

(e) The Geometry Graph Partitioning (GGP) Heuristic

This section deals with a partitioning heuristic based on local search algorithms for Euclidean graphs. The element mesh (or tensor-grid) of a 2D or 3D domain is an Euclidean graph, with vertices the nodal (or grid) points and links the edges of the elements (or the grid line segments between any two consecutive grid points). The matrix and domain decomposition methods require quasi-uniform partitionings of the spatial domain with a minimum diameter. A partitioning heuristic for arbitrary graphs, like KL's partitioning heuristic, is unable to use the geometric properties of Euclidean graphs and deliver partitionings required for matrix and domain decomposition methods. In [CHENHR89] we presented the *geometry graph partitioning* (GGP) heuristic which is an extension of the KL heuristic. The GGP heuristic uses the geometrical properties of mesh graphs by using Euclidean metrics and minimizes the diameter of the subdomains, thus it can deliver quasi-uniform partitionings with

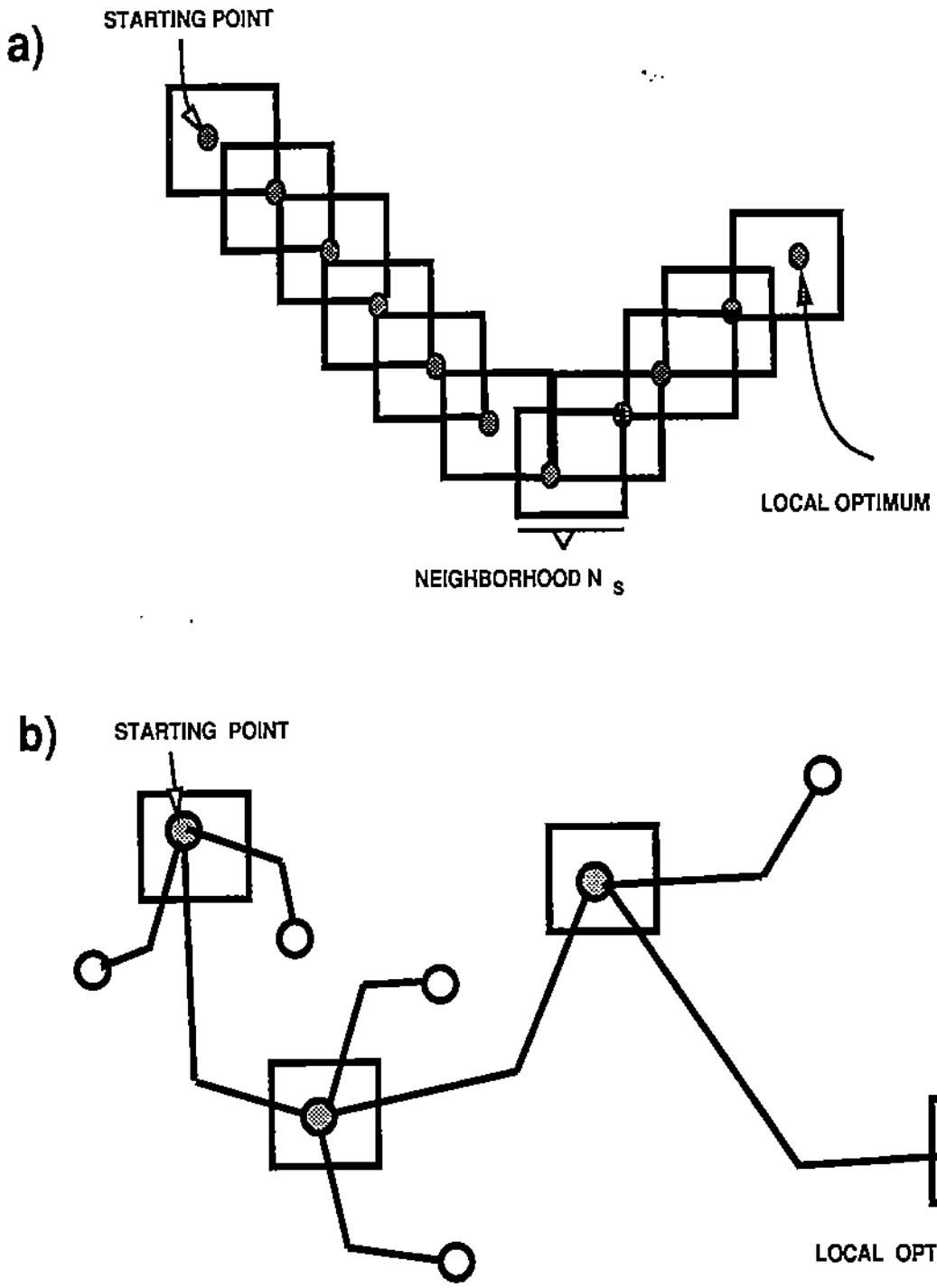


Figure 2.14 (a) Schematic representation of ordinary local search, (b) Schematic representation of variable depth search.

the minimal diameter. Next we outline an improved (in terms of time and space complexity) version of the algorithm presented in [CHENHR89].

The partitioning problem of a discrete PDE domain is transformed into the *graph partitioning* problem of an Euclidean graph (mesh graph). Then the mesh graph is decomposed by the GGP algorithm.

The geometry and the topology of the mesh graph are represented by two *augmented open hash tables*, see Figure 2.15 for a geometric representation of these data structures. Similar data structures are used in other areas like compilers and VLSI design of printed circuit graphs. These data structures guarantee the *linear space* and *quasi-linear time* complexity of the KL and thus the GGP algorithm (see in [FM82] for more details on time complexity.)

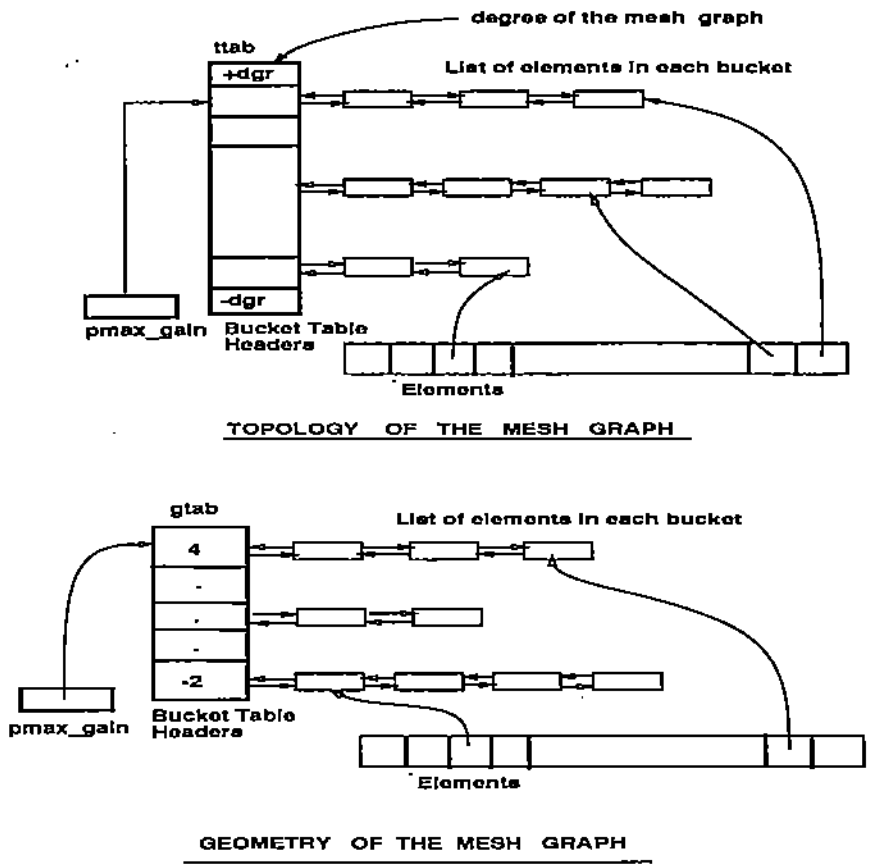


Figure 2.15 Augmented open hash table used to implement the Kernighan_Lin and Geometry Graph Partitioning algorithms.

The GGP heuristic decomposes an element mesh into two subdomains that satisfy to some degree the criteria (i)-(iv) by searching for a sequence of swaps that maximize the following summation :

$$\sum_i profit(i)$$

where

$$profit(i) = \omega_1 f(a_i, b_i) + \omega_2 g(a_i, b_i) \tag{2.6}$$

and

$$f(a_i, b_i) = 2 \sum_{e \in c_{a_i}} \chi(a_i, e) - |c_{a_i}| + 2 \sum_{u \in c_{b_i}} \chi(u, b_i) - |c_{b_i}| - 2\chi(a_i, b_i) \tag{2.7}$$

and

$$g(a_i, b_i) = (\frac{d_{a_i,c_A}}{r_A} - 1) - (\frac{d_{b_i,c_A}}{r_A} - 1) + (\frac{d_{a_i,c_B}}{r_B} - 1) - (\frac{d_{b_i,c_B}}{r_B} - 1) \tag{2.8}$$

where

- $|c_{a_i}|$ and $|c_{b_i}|$ the number of adjacent vertices to the vertices $a_i \in A$, $b_i \in B$ respectively,

- c_A, c_B are the mass center of the subdomains A, B (see Figure 2.16),

- d_{a_i,c_A} and d_{b_i,c_B} are the distances between the elements a_i, b_i and the mass centers c_A, c_B of the subdomains A, B respectively, and

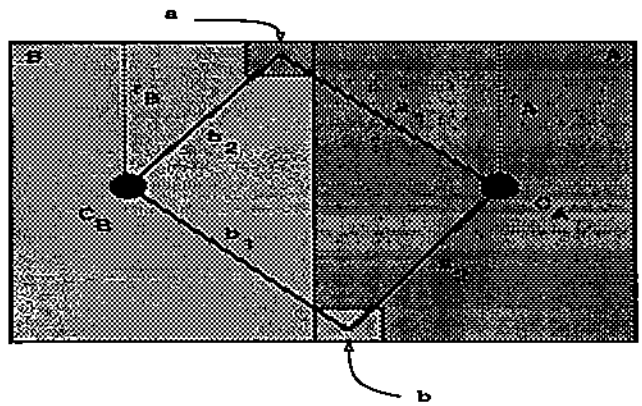
- r_A, r_B are the "ideal" radius of the subdomains A, B.

Algorithm 2.6

begin

```
0. Compute characteristics of the initial partition :
   - Compute mass Centers of the subdomains,
   - Initialize data structure which store the topology
     and geometry of the mesh.
   - Compute the separator of the initial partition.
1. While (either there is a set of pairs S1 for which
         the summation of  f(i) >  0 for each i in S1
         or there is a set of pairs S2 for which the
         distance between the mass centers of the two
         subdomains increases) do
```

begin



Geometric profit from the swap of (a, b) is :

$$\frac{[a_1/r_A - 1] - [a_2/r_A - 1] + [b_1/r_B - 1] - [b_2/r_B - 1]}{\underbrace{>0}\quad\underbrace{<0}\quad\underbrace{>0}\quad\underbrace{<0}}$$

$$>0$$

Figure 2.16 Representation of the Euclidean metrics c_A and c_B, the mass centers of the subdomains A and B, d_{a_i,c_A} and d_{b_i,c_B} the distances between the elements a_i and b_i and the mass centers c_A and c_B of the subdomains A and B, and r_A and r_B the "ideal" radius of the subdomains A and B, for a 2-way partitioning of a quadrilateral mesh.

```
        the Hash Tables)
    begin
      - Set icounter++
      - Store the pair of nodes in a temporary space
        for later use (i.e., X[size++] = node0 and
        Y[size++] = node1)
      - Mark the nodes and delete them for the Hash
        Tables
      - Update the mass centers assuming the swapping
        takes place
      - Compute the new distance, new_dist,  of the
         mass centers of the subdomains
      - if (max_dist < new_dist)
          begin
           max_dist = new_dist;
           S2 = S2  + (node0, node1)
          end
      - Update the interfaces and the Hash Tables
        assuming the swapping takes place
      - Update the pointers of the Hash Tables to
        point at the node with the largest profit
        function (see equation 2.4)
     end
1.3 Compute the maximum subset S1
1.4 if (|S1| > |S2|)
      Swap only the pairs in S1 - S2
    else
      Swap the pairs in  S2
1.5 Compute the new separator
1.6 if (the size of the new separator is larger than the
        size of the separator of minimum size that has
        encountered up to this moment and the maximum
        distance between the mass centers of the
        subdomains is larger than the new distance of the
        subdomains)  then stop
1.7 Update the size of the minimum separator if this
    is necessary
1.8 Update the maximum distance between the mass
    centers of the  subdomains if this is necessary
1.9 Compute the characteristics of the new partition :
     - Compute the mass Centers of the subdomains,
```

```
        - Update the data structure that used to store the topology
          and geometry of the mesh.
        - Compute the separator of the new partition.
    end
end
```

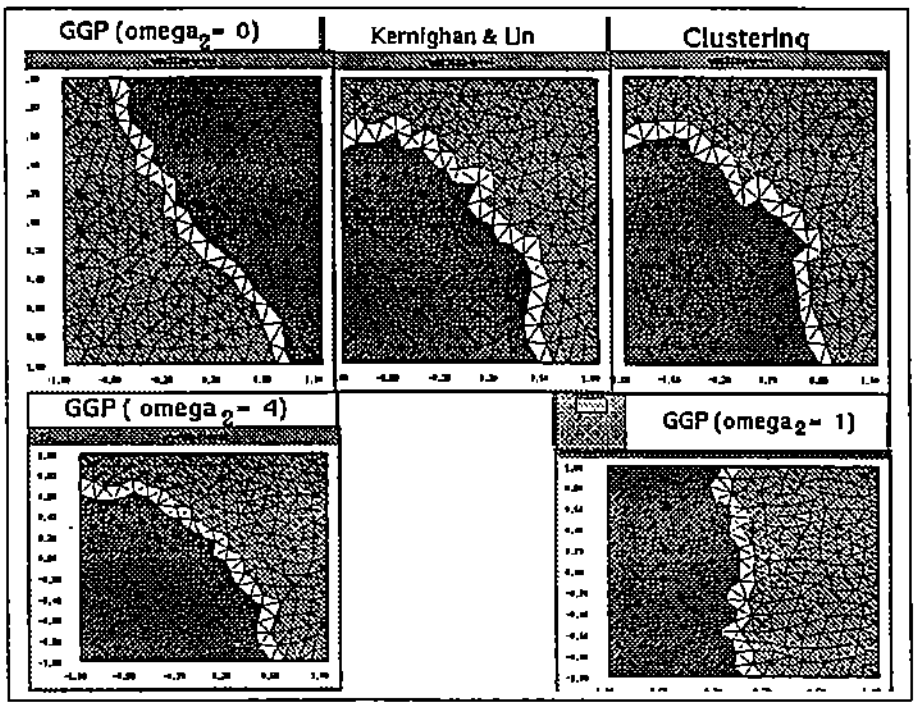


Figure 2.17 The 2-way partitioning of the KL and the GGP algorithm of the initial partitioning by CM_Clust. Different values of ω_2 and $\omega_1 = 1$ have been used.

The GGP heuristic is compared experimentally with the KL partitioning heuristic and consistently returns, within a smaller time interval, partitionings whose separators are smaller in cardinality. Figure 2.18 shows (i) the initial partitioning (top left corner), (ii) the final partitioning by KL algorithm (top center), and (iii) the final partitioning by GGP algorithm (top right corner) and the size of the intermediate separators as a function of the number of swaps (bottom graph). Figure 2.19 shows (i) the final partitioning by KL algorithm (top center) and (ii) the final partitioning by GGP algorithm (to center) and the size of the intermediate separators as a function of the number of swaps (bottom graph) for a random initial partition.

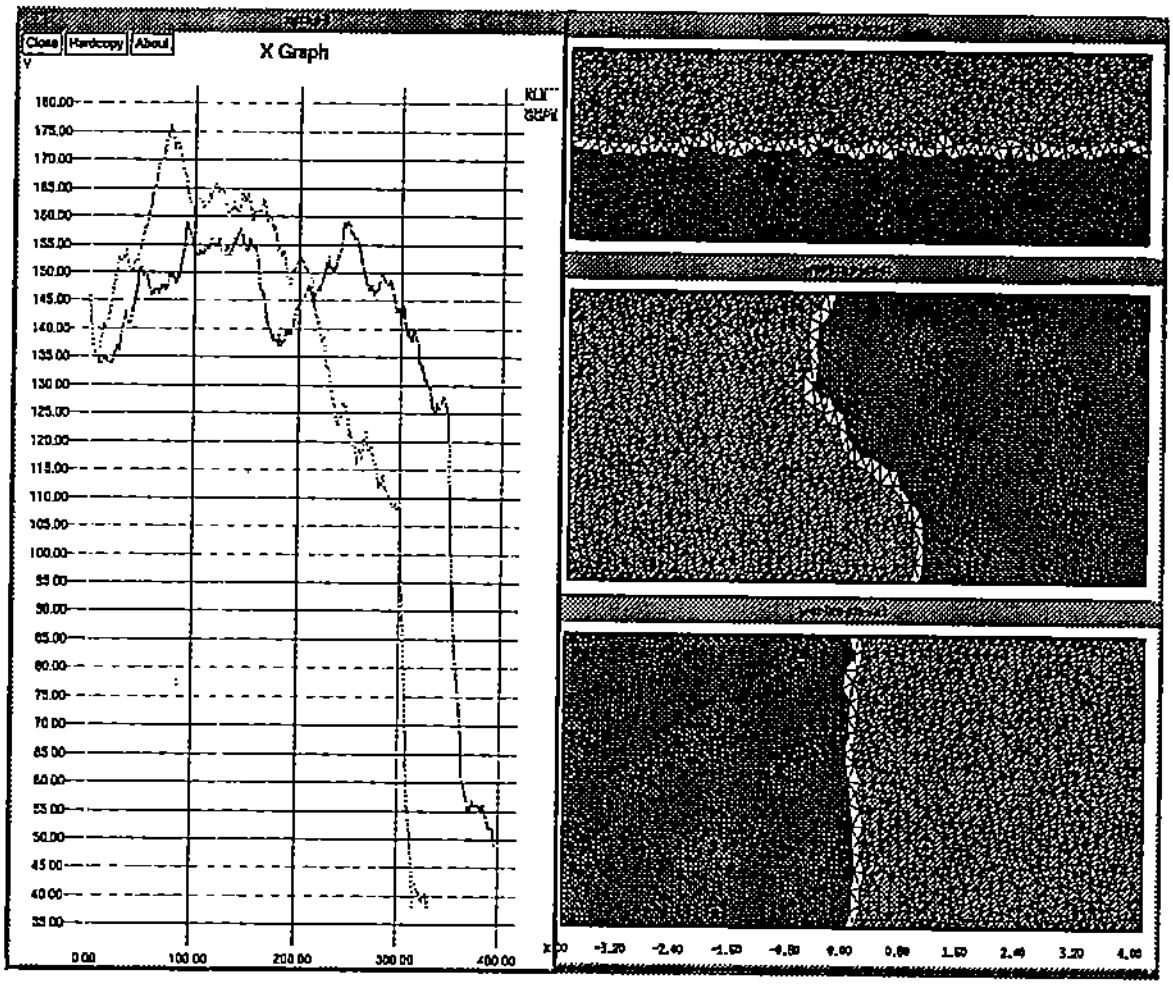


Figure 2.18 2-way partitioning by KL and GGP using as an initial partition the horizontal (1xQ) partition, and the size of the intermediate separators as a function of the number of swaps; x-axis represents the number of swaps and the y-axis represents the value of the objective function (i.e., size of the separator).

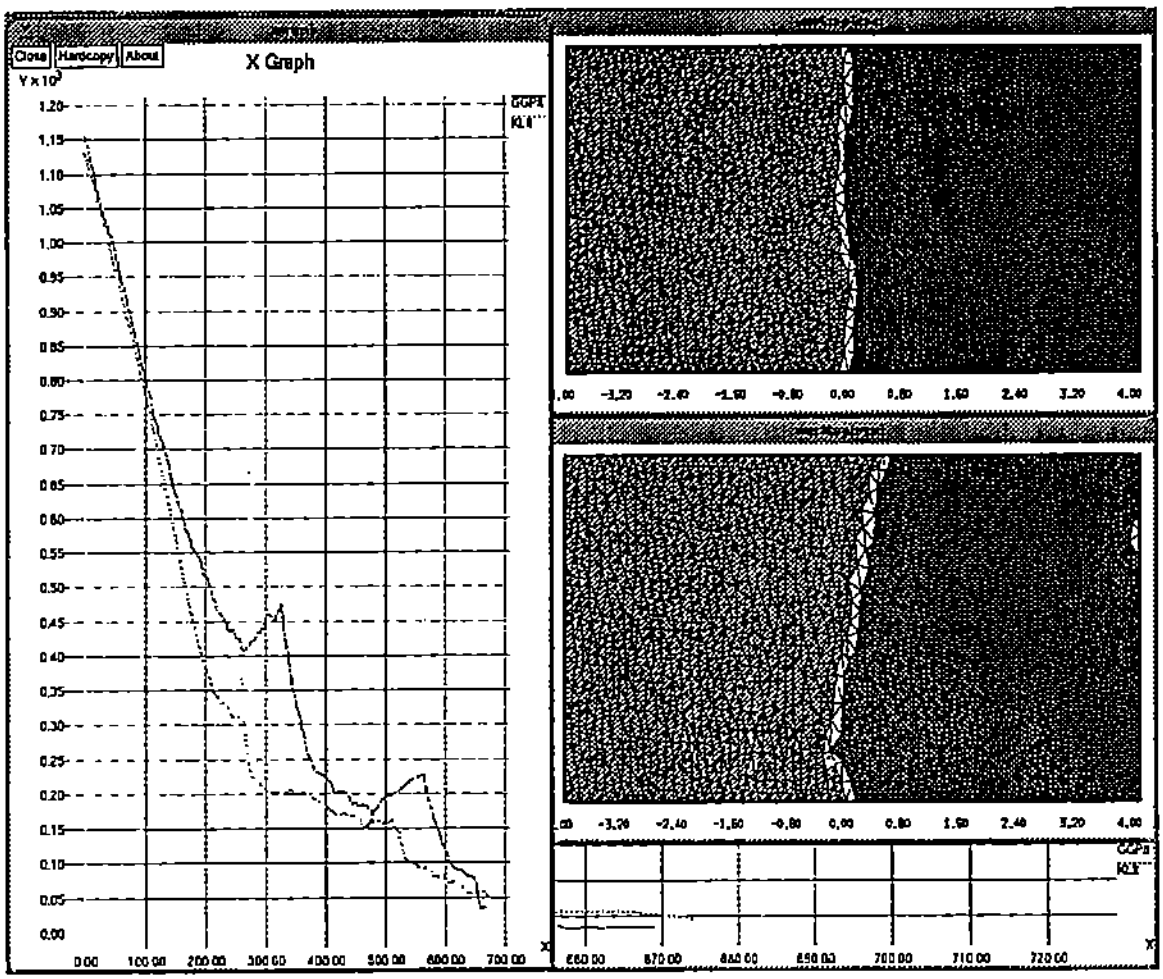


Figure 2.19 2-way partitioning by KL and GGP using a random partitioning as initial partitioning, and the size of the intermediate separators as a function of the number of swaps; x-axis represents the number of swaps and the y-axis represents the value of the objective function (i.e., size of the separator).

2.2.3 Stochastic optimization strategies

Similar to deterministic optimization methods, stochastic methods evaluate the objective function at a sample of points and return as the optimal solution a point that minimizes (or maximizes) the objective function. In the case of the stochastic optimization methods the points are chosen "randomly" from the set of feasible solutions. A naive stochastic optimization method is the one that evaluates the objective function at a randomly chosen feasible point and compares the new cost with the lowest cost encountered. If the new cost is lower, it replaces the current cost and the associated feasible solution is stored. When a certain time limit is exceeded the process is stopped and the last stored feasible solution is returned as the optimal solution. This approach is called *blind random search*

A feasible point often can be generated from another feasible point. Consecutive feasible points in such a generation process are strongly correlated. With such a generation mechanism available one may try to find better optimal feasible solutions by applying local search in neighborhood structures that are similar either to the ones defined in Section 2.1.2 or to the ones defined as a finite set of randomly chosen points from the set of feasible solutions. This approach leads to an other stochastic optimization method, the simulating annealing method. The simulating annealing method is a combination of *iterative improvement* and blind random search with *probabilistic hill climbing* properties.

The simulating annealing scheme is described as follows. First, a randomly chosen feasible point is generated. Then, a change of this point and the evaluation of the objective function at the new point takes place. If the cost of the new point turns out to be lower than the cost of the current point, the new point is stored as the current solution. The process that modifies the current configuration is continued until no improvement is possible. If the maximum allowable time is not yet used, another current configuration is generated randomly and the modification process starts again. The *iterative improvement* of the blind random search in relative short

execution time produces on average much better solutions than the blind search. The disadvantage with these methods is that they might stop in a local minimum. A mechanism that helps these methods to "jump" out of local minima is necessary.

A way to escape from local minima is to introduce an *acceptance function* which is based on the current feasible solution, a newly randomly generated solution, and a random number. The acceptance function is a boolean and decides for the next current optimal solution. It returns true value when the newly generated point either becomes the current solution or it has higher cost. Thus the algorithm can escape from local minima. These algorithms are called *probabilistic hill climbing* algorithms. Probabilistic, because a random number is involved in the decision for the acceptance and "hill climbing" because the objective function may increase during the execution. The performance of these algorithms depends on the definition of the acceptance function. The acceptance function either will remain the same during the execution or its evaluation process can be influenced by some control parameters that can be updated from time to time.

2.3 Allocation heuristics

The problem of allocating computational tasks to target architectures so that the criterion (v) is satisfied has been studied by several authors. In this section we present the proposed allocation techniques that can be applied easily to geometrical data structures.

Bokhari in [Bok81] describes a heuristic formulated on the computational graph, $G_M(V_M, E_M)$, where the vertices V_M are the subdomains that resulted from the partitioning phase of the application. This heuristic starts with an initial assignment of the computational graph vertices to the processors of the interconnection graph and then proceeds with a sequence of pairwise interchanges of the assigned vertices that reduce a predefined *profit* function. The sequence of the pairwise interchanges alternates with probabilistic "jumps" in the case that the search reaches a steady state.

Bokhari concludes that this heuristic may not be suitable for very large computational graphs. In [SE87a] and [SE87b] a number of heuristics have been suggested that couple the partitioning and the allocation phases. These approaches start by generating an initial not "optimal" partitioning and allocation which is improved iteratively. It is easy to observe that these heuristics usually do not produce balanced partitionings for general PDE domains and some of them tend to generate decompositions with disconnected subdomains (see [SE87a] and [SE87b]). Another approach was developed in [FYK87] and it is called the iterative refinement algorithm. Its main idea is to represent the application and the interconnection graphs into a 2-dimensional Euclidean space and then formulate the allocation problem.

2.3.1 Mathematical Preliminaries

In this section we present some mathematical preliminaries to be used for the implementation of the allocation phase. We describe an approach of representing graphs in q-dimensional Euclidean spaces [FYK87] and carry out a new rigorous analysis that suggests a modification which improves the performance of the algorithm (presented in [FYK87]) for mesh graphs.

Let us assume that the connectivity matrix C of G_M is defined by

$$C_{i,j} = \begin{cases} w_{i,j} & \text{if } D_i \text{ and } D_j \text{ are neighboring subdomains,} \\ 0 & \text{otherwise} \end{cases}$$

where $w_{i,j}$ represents the number of interface nodes between the subdomains D_i and D_j. In order to find the "projection" of the G_M graph into the q-dimensional Euclidean space, we formulate and solve the following eigenvalue problem :

$$Bx = \lambda C' x$$

where $B_{i,k} = w_{i,k}/2$ and $C' = \text{diag}\left(\sum_{i=1}^{P} w_{i,\ell}\right)$.

Let λ_k be the kth largest eigenvalue and $x_k = (x_{1,k}, \ldots, x_{pk})^t$ the corresponding eigenvector. Then we define by $u_j = (x_{j,1}, x_{j,2}, \ldots, x_{j,q})^t$ the vector formed by the jth

where $B_{i,k} = w_{i,k}/2$ and $C' = \text{diag}\left(\sum_{i=1}^{P} w_{i,\ell}\right)$.

Let λ_k be the kth largest eigenvalue and $x_k = (x_{1,k}, \ldots, x_{pk})^t$ the corresponding eigenvector. Then we define by $u_j = (x_{j,1}, x_{j,2}, \ldots, x_{j,q})^t$ the vector formed by the jth components of the q eigenvectors $\{x_i\}_{i=1}^{q}$. The $\{u_i\}_{i=1}^{q}$ vectors are called *vertex vectors* in the q-dimensional Euclidean space. For example, using the two largest eigenvalues and the corresponding eigenvectors, we find the representation of the graph vertices in the 2D-Euclidean space [FYSK84] by using the vertex vectors $(x_{j,1}, x_{j,2})$, $j = 1, \ldots, P$.

Next, we prove that this representation which is based on the largest eigenvector is not suitable for a geometry based mapping since the components of the eigenvector corresponding to the largest eigenvalue are equal to 1. Thus all vertex vectors lie on the line $x = 1.0$, which makes the solution of the assignment problem difficult. An alternative way to construct the 2D-Euclidean representation of a graph is to select the eigenvectors of the second and third largest eigenvalues. In the case of graphs with P vertices, this will lead to the following 2D-Euclidean representation :

$$(x_{j,2}, x_{j,3}) \quad j = 1, \ldots, P.$$

The nature of the largest eigenvalue and corresponding eigenvector for the mesh graphs G_M is described by the following assertions:

Lemma 1. If $R = C'^{-1}B = (R_{i,j})$ then $\sum_{j=1}^{P} R_{i,j} = \frac{1}{2}$ for $i = 1, 2, \ldots, P$.

Proof. From the definition of C' and B we observe that $C'_{i,i} = 2\sum_{j=1}^{P} B_{i,j}$, $R_{i,j} = \frac{1}{C'_{i,i}} B_{i,j}$. Thus $\sum_{j=1}^{P} R_{i,j} = \frac{\sum_{j=1}^{P} B_{i,j}}{2\sum_{j=1}^{P} B_{i,j}} = \frac{1}{2}$ for all $i = 1, \ldots, P$.

Lemma 2. The matrix R is non-negative irreducible matrix.

Proof. The non-negative property of the matrix R is a direct consequence of the fact that the matrices B and C' are non-negative. The matrix B is the adjacency matrix of the graph that corresponds to the partitioning of the finite element mesh Ω_h of a simply connected domain Ω. Hence, G_M is a strongly connected graph since for any pair of nodes there exists a path that connects the nodes. This implies that B

Lemma 3 [Var62]. If A is non-negative irreducible $n \times n$ matrix with

$$\sum_{j=1}^{P} A_{i,j} = const., \text{ for all } i = 1, 2, \ldots, n$$

then the spectral radius $\rho(A) = const.$

From the above lemmas, we can easily conclude that the spectral radius of $\rho(C'^{-1}B) = \frac{1}{2}$ and the corresponding eigenvector is $[1, \ldots, 1]^t$.

2.3.2 Geometry based allocation (GBA) strategies

In this section we present two classes of heuristics for the allocation problem : implicit and explicit heuristics. In both cases, we assume that the PDE computation is decomposed by means of decomposing the geometry of its domain into balanced subdomains. We formulate the mesh decomposition graph $G_M(V_M, E_M)$ by viewing the subdomains as the vertices (V_M) of the graph G_M and the connections between neighboring subdomains as the edges (E_M) of the graph G_M. In Figure 2.20 we show the geometry based partition of the PDE domain, which is the output of the GGP algorithm and its dual graph (see Figure 2.21 for the graph of this partition).

Our decomposition techniques allow us to set *a priori* the number of desired subdomains and thus, we can assume that the sizes of the two graphs are equal ($|V_M| = |V_A| = \mathrm{P}$). For general graphs, the assignment problem is equivalent to the minimization of the *cost functions* (2.3) or (2.4). Unfortunately these optimization problems are known to be NP-Complete problems. Thus, we are forced to devise "fast" heuristics. Next, we describe some *explicit* techniques for minimizing the cost function (2.3) and (2.4) and a number of *implicit* techniques which are based on three sets of cost functions.

(a) Implicit GBA Strategies

The idea of an implicit GBA strategy is to project the application and processor interconnection graphs into a simpler space, such as the 2D Euclidean space, and solve the assignment problem on the projected space. This is accomplished by starting with

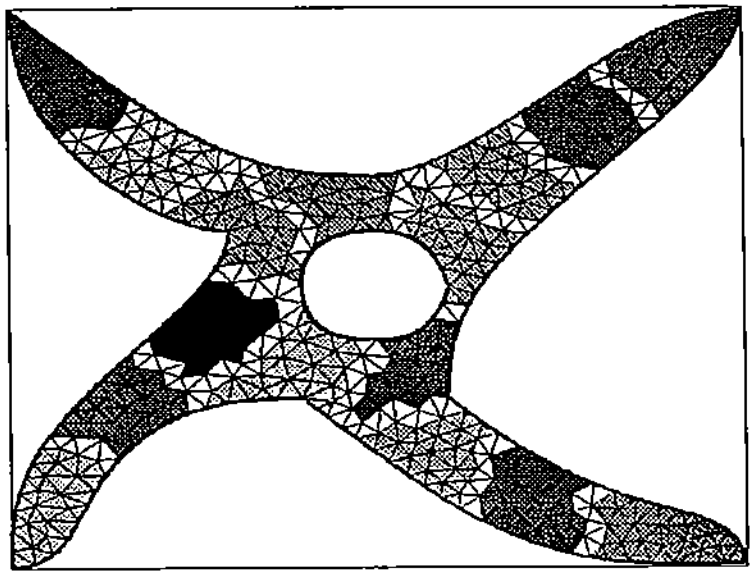


Figure 2.20 16-way partitioning of a spatial domain by the hybrid partitioning heuristic.

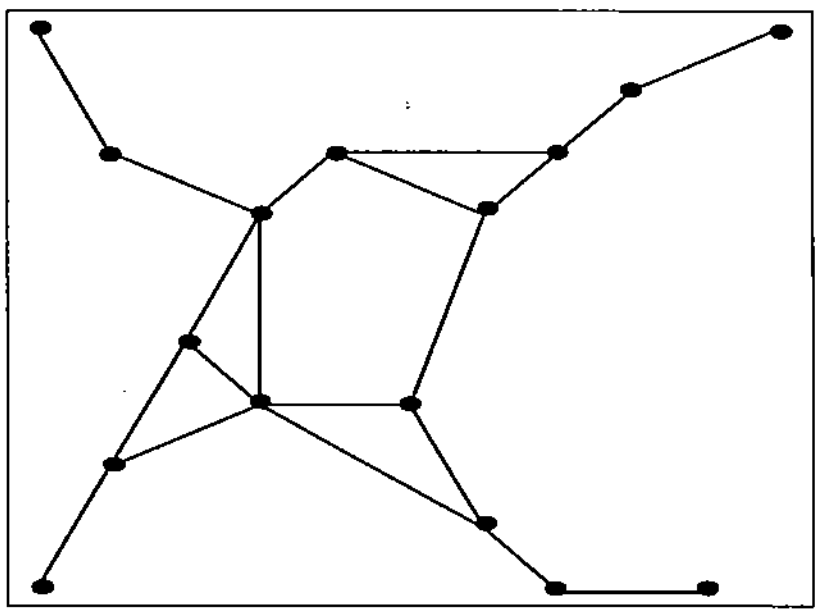


Figure 2.21 The dual graph G_M associated to the partitioning of Figure 2.20.

an initial assignment and then by using iterative refinement techniques to improve on it. There are several phases to this technique as follows:

Phase 1: Representation of the mesh $G_M(V_M, E_M)$ into a 2D-Euclidean space.

A simple way of representing the mesh decomposition $G_M(V_M, E_M)$ into a 2D-Euclidean space is to associate with each node of G_M the coordinates of the mass center of the subdomain D_i, (x_i^M, y_i^M). This is a natural way of representing the nodes of a planar 2-dimensional graph. As a further comment, we mention that a generalized way of representing *general* (non-planar) graphs onto an n-Euclidean space has been presented in [FYSK84]. The idea is to represent the graph by a collection of points in the n-dimensional Euclidean space, such that the distance between vertices reflects the weights of the edges between the vertices of the original graph. Thus vertices of the graph that are connected by edges with large weights are projected to nearby points in the Euclidean space. Then the graph on the Euclidean space reflects the connectivity properties of the original graph. The general graph representation problem is achieved by the solution of a generalized eigenvalue problem which is described in the mathematical preliminaries (Section 3.3).

The experiments we have conducted are concerned with a planar G_M graph. This is always the case when the PDE domain is 2-dimensional and a geometry based decomposition method is used to obtain G_M. A more general G_M graph is obtained when the dimension of the PDE domain is three or more.

Phase 2: Representation of the system graph into a 2D-Euclidean space.

The same approach can be used for the representation of the interconnection graph $G_A(V_A, E_A)$ of the architecture. This graph is projected into the same 2D-Euclidean space used for the G_M graph. The G_A graph depends on the architecture of the machine and thus, in general, it is non-planar. Hence, the generalized method in [FYSK84] can be used.

In our experiments, we have utilized the nCUBE-6400 machine. For this architecture it is sufficient to project a 2D-grid onto the common 2D-Euclidean space since

the 2D-grid graph can be embedded into the hypercube graph [SS88]. Actually one can use the [0, 1]x[0, 1] as the Euclidean space and then assign coordinates to the nodes of the 2D-grid graph. The grid point (i,j) of a 2D-graph with $N1$ vertices along the x-axis and $N2$ vertices along the y-axis can be represented in the Euclidean space by the $(x^A_{m(i)} = i/N1, y^A_{m(i)} = j/N2)$. Therefore, the problem is simplified when the machine architecture is such that a grid can be embedded into it. Moreover, the geometry based decomposition provides a planar two dimensional graph with number of vertices equal to the number of grid nodes. Thus, the allocation problem is simplified into a planar assignment problem. Next, in the phases 3 and 4, we look into the planar assignment problem.

Phase 3: The planar assignment of the $E(G_M)$ onto $E(G_A)$.

Let us represent the projected G_A, G_M graphs into the 2D-Euclidean space by $E(G_A)$, $E(G_M)$ respectively. Figure 2.22 shows the projections $E(G_A)$, $E(G_M)$ of the graphs into the 2D-Euclidean space.

The allocation of $E(G_M)$ onto $E(G_A)$ is equivalent to the minimization of one of the following objective functions:

let

$$d(i, m(j)) = (x^M_i - x^A_{m(j)})^2 + (y^M_i - y^A_{m(j)})^2$$

1.

$$d_{1,k} = \min_m \sum_{i,j=1}^{P} d(i, m(j))^{1/k} \quad for\ k\ =\ 1,\ 2 \tag{2.9}$$

2.

$$d_2 = \text{Rectilinear (Manhattan) distance} \tag{2.10}$$

3.

$$d_3 = \min_m \{ \max_{i=1,\ldots P} \sum_{(x^M_j, y^M_j) \in C_{(x^M_i, y^M_i)}} d(i, m(j))^{1/k} \} \tag{2.11}$$

where $C_{(x^M_i, y^M_i)} = \{(x^M_j, y^M_j) \in E(G_M)\ \ and\ \ the\ \ subdomain\ \ D_j \in N(D_i)\}$.

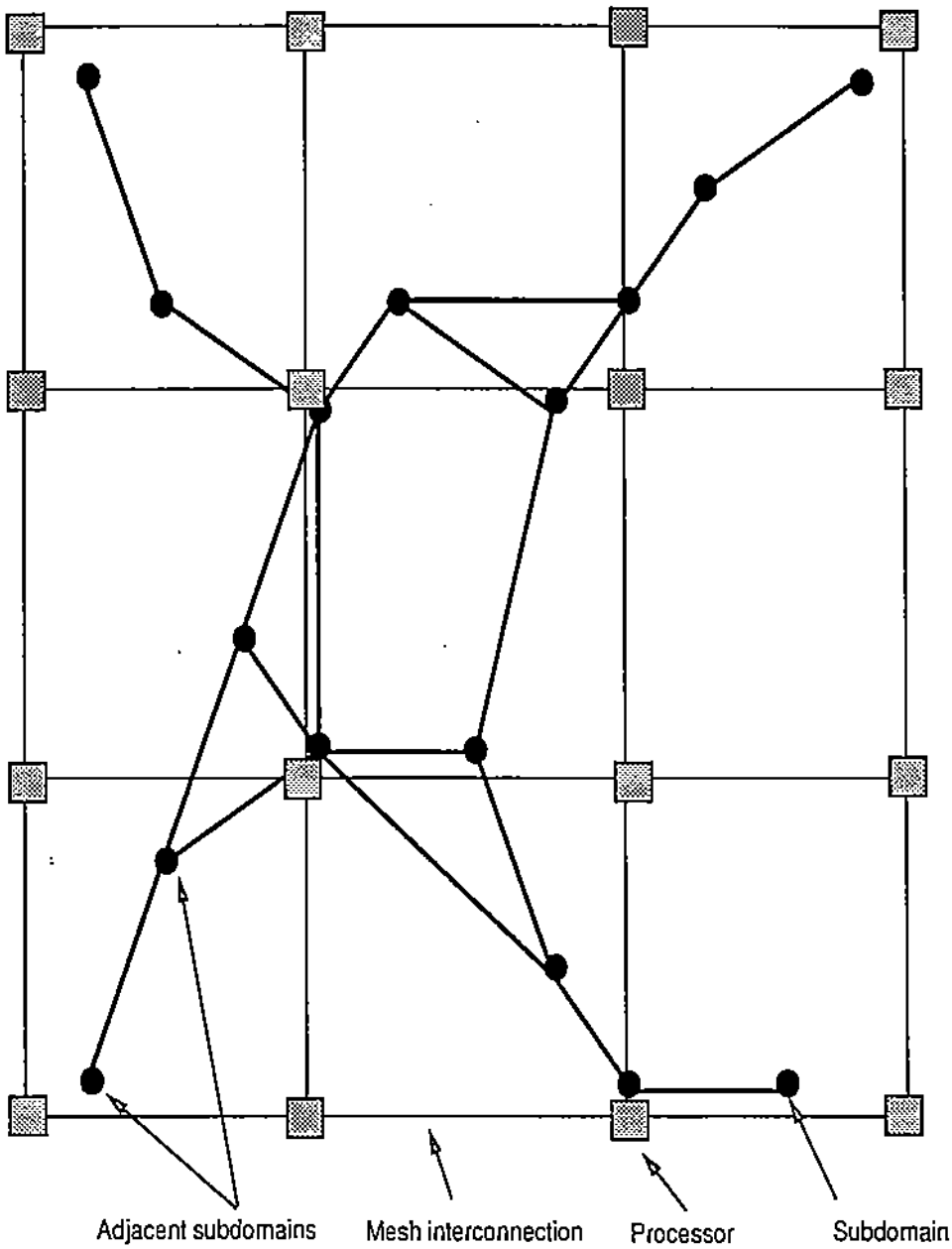


Figure 2.22 Projections of $G_A(V_A, E_A)$ and $G_M(V_M, E_M)$ graphs into 2D-Euclidean space.

The minimization of the first two functions results in the allocation of neighbor subdomains onto neighbor processors, while the third objective function attempts to sort the subdomains with respect to communication overhead and then allocate them and their neighbors according to this sorting order. Farhat in [Far89] independently developed similar allocation approach based on the cost function d_3.

Let us denote by *Fucun* the allocation technique developed in [FYK87] and by *GBA_d* the geometry based allocation, which is achieved by minimizing one of the distance objective functions as defined in (2.12)-(2.14). The geometry based allocation also implicitly assumes a geometry based partition. In our case five such partitions are used, named after the algorithm used for their implementation, *CM_Clust, Rec_Bisec, RxQ, 1xQ (R=1) and Hybrid*, where

CM_Clust	:	A clustering techniques based on the ordering of nodal
	:	points using the algorithm 2.3, section 2.2.1.
Rec_Bisec	:	A recursive bisection approach based on 2-way
	:	rooted level structure algorithm, section 2.2.1.
RxQ	:	Block partitioning along the x and y direction, section 2.2.1.
1xQ	:	Strip partitioning along x or y direction, section 2.2.1.
Hybrid	:	Recursive bisection using the GGP heuristic, section 2.2.2, whose
	:	initial 2-way partitioning is determined by CM_Clust.

So far in phase 3, we provided an allocation of the G_M graph onto the G_A graph, which we regard as an initial assignment. In phase 4, we try to improve on it by means of an iterative improvement technique as follows.

Phase 4: The iterative improvement of the initial assignment.

The last phase of the mapping process is the most time consuming phase and depends strongly on the success of the initial mapping. We have implemented an instance of the iterative refinement approach proposed by Goto in [Got81], which is similar to the one developed by Kernighan and Lin in [KL70]. A high level description of this procedure is given in the form of an algorithm whose input parameters include

the number of the subdomains to be interchanged during the search and the distance ϵ which is the "neighbor" distance among two allocated subdomains.

Algorithm 2.7

```
iter_impr(lo, e, initial_mapping)
*
* lo        is the number of subdomains to be interchanged
*           during the search
*
* N(A, e)   is the set of all subdomains X so that
*           || m(X) - m(A) || < e
*
 begin
1. l := 2
2. Compute initial communication cost, using one of
   the quality functions (2.3) or (2.4)
3. Choose a module to start the iteration, call it
   primal module, and declare it as the A module.
4. Mark A, and Compute the  e-neighborhood of A.
5.  for  each  X of N(A,e)   do
     Compute the reduction in the communication
     cost assuming that the interchange (A, X)
     takes place.
    endfor
6. Compute the maximum reduction in the communication,
   let us call it  CRmax
7.  if  CRmax < 0   then
     l := l + 1
      if l < lo  then
        continue_search(A, l, lo)
      else
        Choose the next non_marked primal module, and
        repeat from step 2. if all primal modules has
        been exhausted then stop
    endif
    else
     Interchange (A, Xo),  where  Xo corresponds to
     the maximum reduction in the cost function (2.3)
     or (2.4) and Update the  new_total_communication.
    endif
8.  Repeat from 2   until all modules have been exhausted.
 end
```

The iterative improvement algorithm was applied on the initial allocation as presented in phase 3. The improvement we obtained for our example was of the order of 10% reduction of the cost functions (2.3) and (2.4). This is partly due to the fact that the problems we are solving are small (G_M graph nodes G_A graph nodes) and thus the techniques of phase 3 are adequate. We expect that Phase 4 will give significant improvement for large problems, i.e., $|G_M| \geq 1000$.

(b) Explicit Heuristics

We have already observed that the search of the optimum solution of (2.3) and (2.4) is computationally very intensive and thus impractical for $P > 15$. Instead various approximation methods have been proposed. One efficient algorithm for solving the quadratic assignment problem (QAP) (for its formulation see 2.3) has been suggested in [Wes83]. Another approach is to approximate (2.3) by a linear assignment problem (LAP) based on the cost matrix C of QAP. The LAP can be stated as seeking a permutation $m = \{n_1, \ldots, n_P\}$ on the integers $\{m(D_1), \ldots, m^{-1}(D_P)\}$ that minimizes

$$l(m) = \sum_{1 \leq i \leq P} C_{D_i, m(D_i)}.$$

To solve the LAP, we have implemented the algorithm suggested in [CT80]. Figure 2.23 depicts the evaluation of the QAP and the LAP approximate allocations found, as measured by the cost function (2.3). For the generation of these data, we have used the Hybrid partitioning on a rectangular domain.

2.4 Scheduling heuristics

The parallel iterative PDE solvers require two kinds of synchronization, *Global* and *Local* synchronization (LS). Global synchronization is required for things like evaluating stopping criteria or accelerating the convergence. Its execution time can be minimized by (i) designing new numerical algorithms and (ii) using more efficient global synchronization software primitives supported by faster hardware. Local

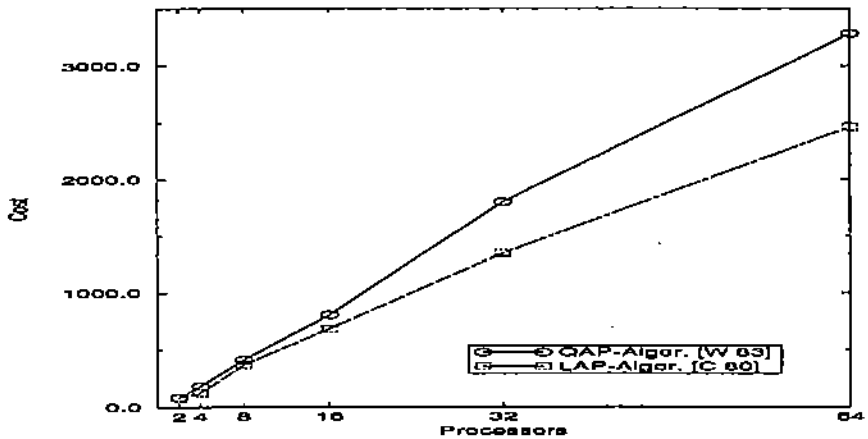


Figure 2.23 Evaluation of the solutions, for the allocation problem, obtained by QAP algorithm [Wes83] and LAP algorithm [CT80] for the objective function (2.3).

synchronization is required for the exchange of local data between neighboring subdomains of a partition. The time for the Local Synchronization Segment (LSS) (see section 2.1) can be minimized by finding (i) nearly optimal partitionings and allocations of the computation onto the parallel system and (ii) efficient scheduling of the messages from the source to destinations (message scheduling) in order to minimize edge or bus contentions.

We consider the impact of LSS time on the performance of parallel PDE iterative solvers for a model problem (Poisson problem defined on the unit square) using two MIMD architectures (hypercube and mesh). First, we focus on minimizing the LSS time by finding "optimal" schedules for the messages to eliminate the edge-contentions in hypercube and mesh circuit switched networks.

2.4.1 k-Wave static local message scheduling

Bokhari in [Bok90] and we have observed that unwise use of the parallel hardware resources can increase the communication time by a factor of more than seven. For a circuit switched machine like the nCUBE-6400, the edge contention (i.e., the sharing of a communication link by two or more paths) increases the communication overhead. For a packet switched machine like CM-5 the node contention (i.e., the sharing of a

node by a number of paths larger than the number of I/O channels of the network interface chip) will increase the communication time.

The edge contention problem on a circuit switched interconnection (either hypercube or mesh) can be resolved by a two step scheduling mechanism of the local messages. For a given partitioning graph we compute the chromatic number, say k, of the graph and group its vertices into k subgroups and then, we assign to each group a communication pattern for its local messages. Different processors that belong into different subgroups have different communication patterns (i.e., orderings of the local messages). The k different communication patterns are designed in a way that for any pair of local messages of neighbor processors which are in different subgroups, there is a difference of 360/k degrees in direction (on a 2D-mesh interconnection). We call this k-Wave static local scheduling. Figure 2.25 depicts this scheduling algorithm for the Ext_R×Q (Figure 2.24.) partitioning of the unit square which is mapped on a 2D-mesh interconnection. Ext_R×Q partitioning algorithm is the same as R×Q but for each subdomain the algorithm uncouples its northeast and southwest neighbor subdomains.

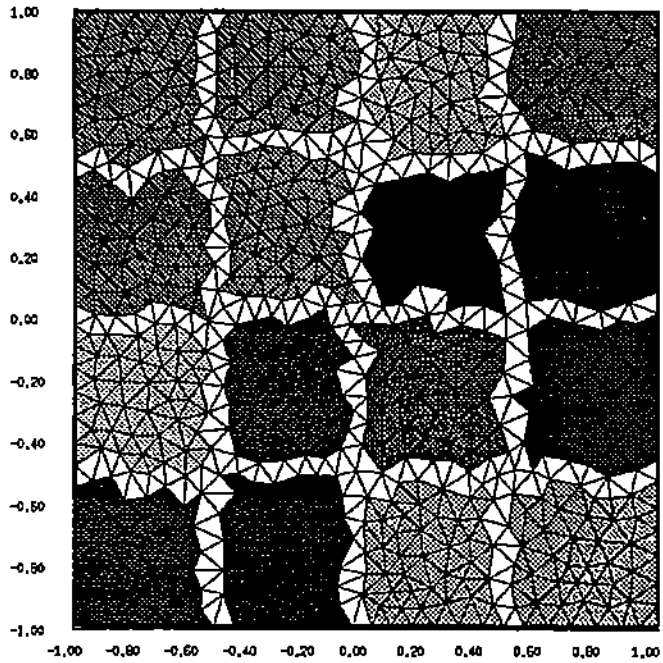


Figure 2.24 Ext_R×Q (with R = 4 and Q = 4) partitioning of a 2 dimensional square domain.

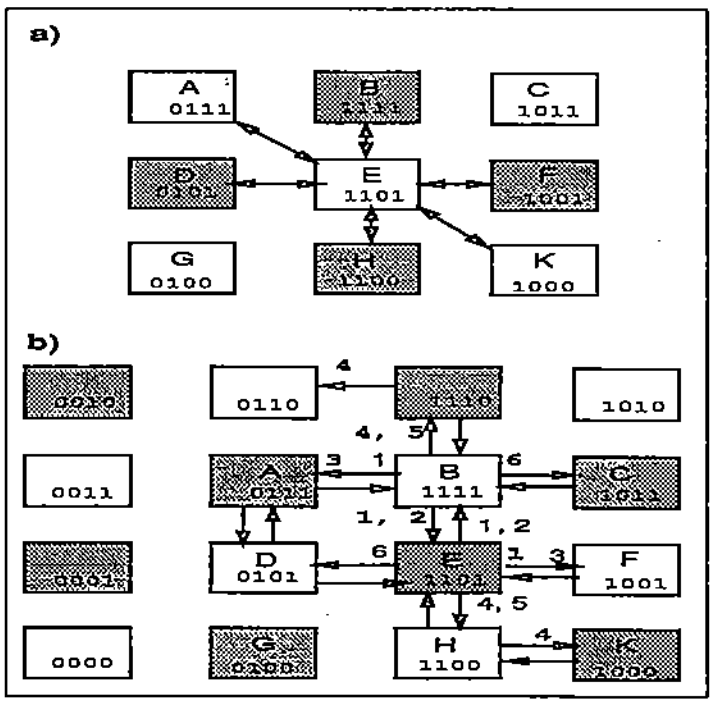


Figure 2.25 2-Wave scheduling mechanism of local messages for the partition of Figure 2.24.

3. PERFORMANCE EVALUATION OF GEOMETRY BASED MAPPING TECHNIQUES

In this chapter we present the performance evaluation of the mapping techniques described in chapter 2. First, the performance is measured in terms of the satisfiability of the criteria (i) to (iv) through which we have defined the three phases of the mapping process. In the case of criteria (i) and (ii) we evaluate the *interface length* and *communication cost functions*, as defined by cost functions (2.3) - (2.5), at the computed feasible mapping (i.e., partitioning and allocation). Criteria (iii) and (iv) are evaluated by inspection. The "quality" of the computed mapping with respect to criteria (v) and (vi) is verified indirectly by measuring the *local communication* cost of the underlying computation on the targeting parallel machine. This local communication cost that includes the overhead of synchronization of the computation is defined by the mathematical model (2.5) which is equivalent to the execution time of the algorithmic segment described in Figure 2.2. This segment implements the communication requirements of a domain decomposition iterative PDE solver. Finally, the performance of the various mappings, consisting of pairs of partitioning and allocation strategies with different message schedulings, is evaluated based on the execution time of several domain decomposition based iterative solvers from the Parallel ELLPACK library [CHK+92] on the nCUBE-6400. Notice that all the parallel iterative solvers used are equivalent to the sequential ones from ITPACK library [KRYG82].

3.1 The nCUBE-6400

The nCUBE-6400 is a hypercube interconnection multiprocessor system [nCU91]. A unique binary ID has been assigned to each processor of the network. Two vertices

in the network called neighbors iff their binary representation differ in exactly one bit. Each processor has its own memory and works independently from the others. Processors exchange data by sending messages to each other. The exchange of messages is based on a circuit switching logic. When two nodes have to communicate, a fixed path is set up between them. The message flow through this path is without interapting the intervening processors. The path between the two nodes (source and destination) is established by a fixed routing strategy : the source node sends an address packet of 32 bits to a channel. The address is passed from node to node. Each processor compares his own binary ID with the address packet and if the processor is the destination, the path has been established, otherwise, it forwards the packet to his neighbor processor whose binary ID most closely matches the binary ID of the destination. The time (in μsec) to communicate a zero byte in a network without edge contention is :

$$T_{initial} = 2.84 \times d + 132.87 \tag{3.1}$$

where d is the Hamming distance.

3.2 Evaluation of Partitioning Heuristics

In this section we compare five partitioning strategies of a 2-dimensional triangular finite element meshes with respect to the various criteria used for their formulation. These strategies are:

CM_Clust	:	A clustering techniques based on the ordering of nodal points using the algorithm 2.3, section 2.2.1.
Rec_Bisec	:	A recursive bisection approach based on 2-way rooted level structure algorithm, section 2.2.1.
RxQ	:	Block partitioning along the x and y direction, section 2.2.1.
1xQ	:	Strip partitioning along x or y direction, section 2.2.1.
Hybrid	:	Recursive bisection using the GGP heuristic, section 2.2.2, whose initial 2-way partitioning is determined by CM_Clust.

Their evaluation is based on the following indicators : a) the percentage of the total number of interface points with respect to the total number of the points of the mesh, b) the percentage of the number of interface points with respect to total number of points per subdomain, c) the average connectivity of the subdomains, d) the maximum connectivity of the subdomains, and e) the execution time of the partitioning algorithms.

For the performance evaluation of the above partitioning strategies, we have considered a PDE problem based on the Poisson operator and Dirichlet boundary conditions with the domain of definition depicted in Figure 1.1. For its discretization we used the finite element method based on linear triangular elements. The mesh used consists of 18890 elements and 9880 nodes. The system of finite element equations was solved by a parallel version of the Jacobi-SI method [CHK+92]. Throughout we refer to this test case as BENCH1.

Figures 3.1 and 3.2 depict the percentage of interface points for the entire domain and for a subdomain respectively in the case of BENCH1 computation. These data indicate that the Hybrid heuristic solution is the best and the 1xQ the worst. This agrees with the theoretical behavior of these strategies. The 1xQ makes no attempt to minimize the interface length. Figure 3.3 and 3.4 plot the average and maximum connectivity of various decompositions of the triangular mesh of BENCH1 produced by the five strategies. The data indicate that 1xQ and Hybrid heuristics give the best partitionings with respect to this criterion. This was expected since both are designed to minimize this indicator. The connectivity of 1xQ is by construction "small" while the GGP (the second part of the Hybrid) explicitly incorporates this requirement in the profit function. The rest of the strategies make no attempt to minimize this criterion. Figures 3.5 and 3.6 present the execution time of these heuristics on a SUN Sparc station 2. It is not a surprise that the "naive" ones are the cheapest. It is easy to realize that the computations involved in determining the partitioning of a mesh are inherited parallel. No attempt has been made in this thesis to parallelize them. This will be part of our future research. Finally, the performance of the five

Table 3.1 Performance evaluation of all P-way partitioning methods considered in chapter 2 with respect to the satisfiability of criteria (i) to (iv).

Algorithm	(i) Load Balance	(ii) Interfaces	(iii) Degree	(iv) Connectivity
CM_Clust.	√			
Rec_Bisec.	√			
RxQ	√	√	√	
1xQ	√		√	
Hybrid	√	√	√	√

partitioning heuristics considered with respect to satisfiability of the criteria (i) to (vi) is summarized in Table 3.1. In the case of criteria (iii) and (iv) the satisfiability is verified by inspection while the criterion (ii) is considered to be satisfied if the percentage of the overall interface length is below 50 % of the total mesh points.

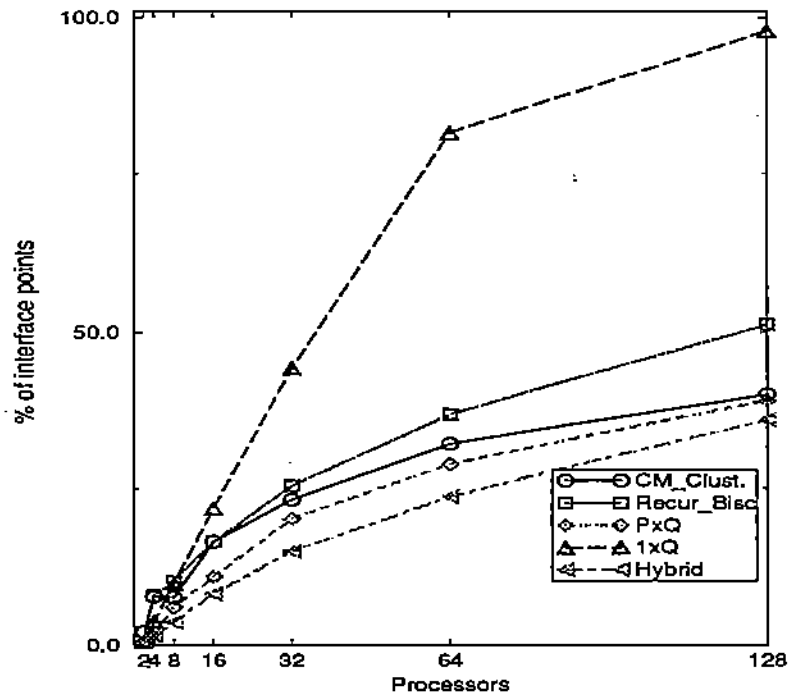


Figure 3.1 The percentage of interface nodal points for the BENCH1 computation for the five different partitioning schemes listed in section 3.2.

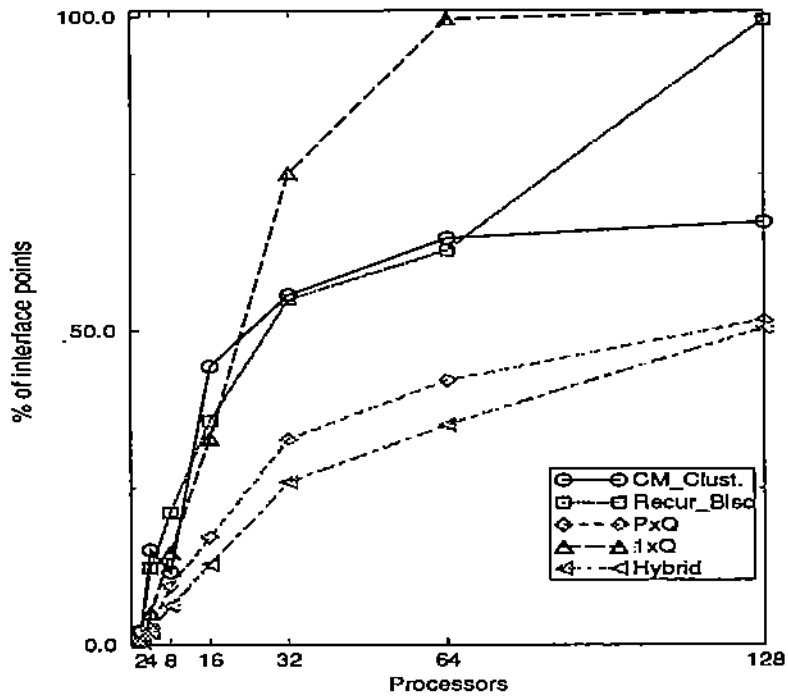


Figure 3.2 The percentage of interface nodal points per subdomain for the BENCH1 computation for the five partitioning schemes listed in section 3.2.

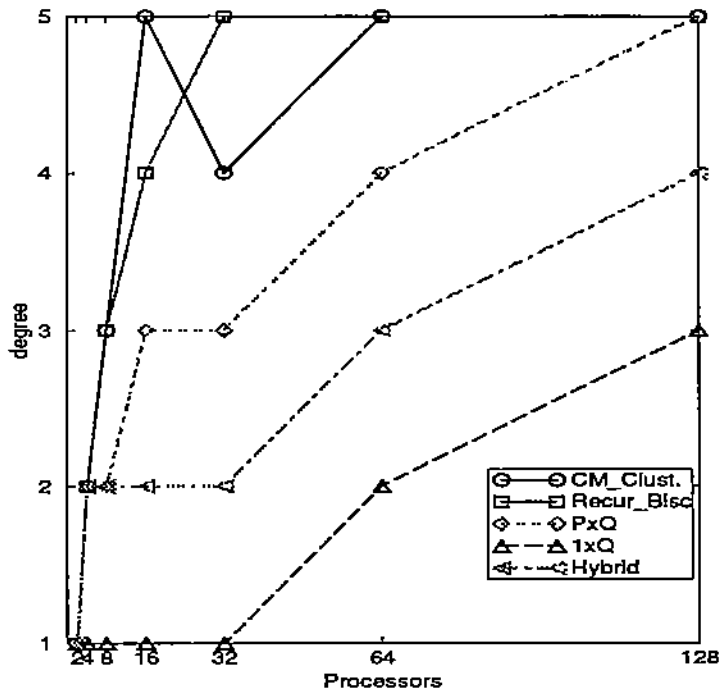


Figure 3.3 Average degree of the decomposition graph produced by the five partitioning schemes listed in section 3.2 for BENCH1.

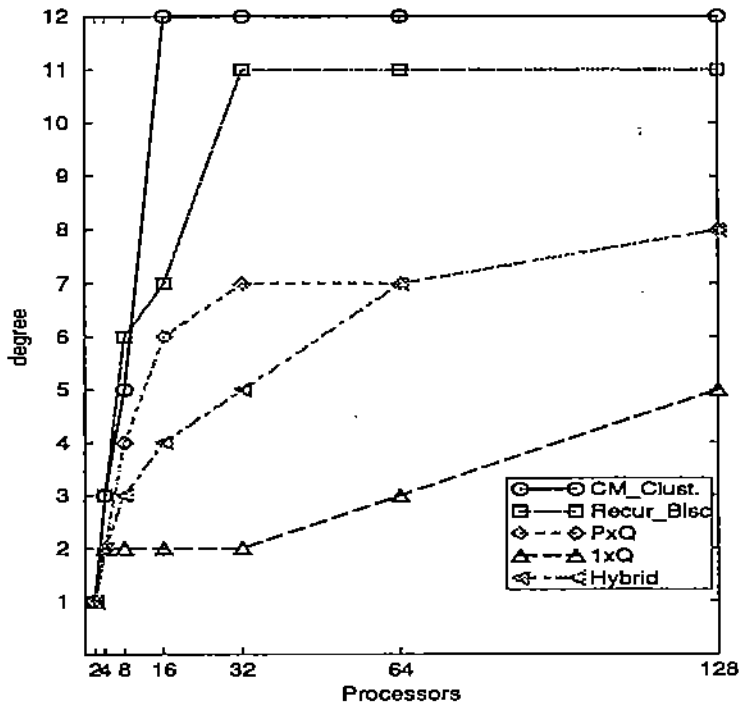


Figure 3.4 Maximum degree of the decomposition graph produced by the five partitioning schemes listed in section 3.2 for BENCH1.

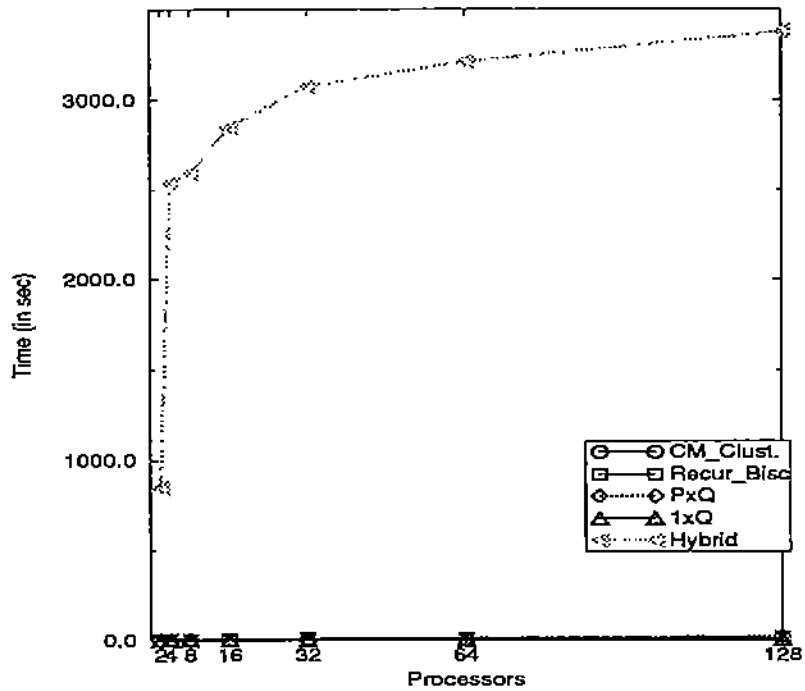


Figure 3.5 Total execution time (in sec) of the five partitioning schemes listed in section 3.2 on SUN Sparc 2 for BENCH1.

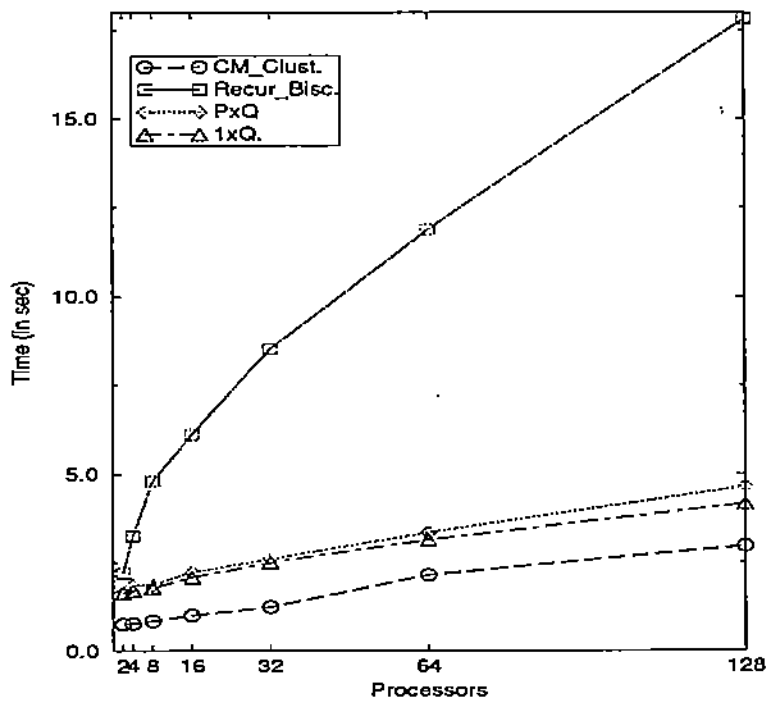


Figure 3.6 Total execution time (in sec) of four fastest partitioning algorithms on SUN Sparc 2 for BENCH1.

3.3 Allocation Heuristics

In this section we measure the impact of four different allocation strategies to the numerical solution of elliptic PDEs on the nCUBE-6400 machine. Specifically, for the benchmark BENCH1 computation specified in section 3.2, we keep the partitioning fixed and vary the allocation strategy and measure the cost of local communication or synchronization as defined by relation (2.0). We compute the elapse time of each Jacobi-SI iteration, the local communication cost by measuring the execution time of the algorithmic segment which implements it (see Figure 2.2), and the percentage of the local communication overhead over the total elapse time.
The five allocation strategies considered are:

GBA_$d_{1,1}$	: Geometry Based Allocation, minimizing equation (2.12) for k=1.
QAP	: Heuristic for QAP presented in [Wes 83].
Shift	: Heuristic mapping the subdomain D_i to processor $i-1$.
Random	: Heuristic mapping subdomains to processors at random.

Tables 3.2 and 3.3 measure the "quality" of the different allocation strategies using the objective functions (2.3) and (2.4) respectively. Note that the mapping based on the QAP allocation and the Hybrid partitioning strategies returns the best allocation with respect to both objective functions. The behavior of GBA_$d_{1,1}$ and Shift is almost equivalent under all partitionings considered. According to Table 3.4 the QAP is the slowest of all. Tables 3.5 to 3.7 indicate the performance of the domain decomposition based Jacobi-SI solver under different mapping strategies of the finite element mesh. In these experiments the scheduling of the communicating data is done according to the first-in first-out (FIFO) mechanism. The data in Tables 3.6 to 3.7 indicate the magnitude of the various overheads due to the parallelization of the Jacobi-SI solver. These data suggest that the allocation strategies have no impact in this computation while the partitioning affects the execution time of the underlying computation. We believe that this effect will be magnified for much larger meshes

and machine configurations. We plan to explore this issue further as part of our future research activities. It appears that for moderate size problems and machine configurations the naive mapping strategies are effective. In our future research we will attempt to verify this behavior for massively parallel systems (P $\geq$ 1000).

Table 3.2 The performance evaluation of 20 mappings based on the combination of four different allocation strategies and five partitionings of the mesh Ω_h (used in BENCH1) with respect to satisfiability of criterion (v) as it is measured by the quadratic function $\sum_{i=1}^{P} \sum_{j=1}^{P} c(D_i, D_j) d(m(D_i), m(D_j))$.

Algorithms	Hybrid	CM_Clust.	Rec_Bisec.	PxQ	1xQ
$GBA\_d_{1,1}$	12820	39171	38587	17751	26271
QAP	8828	28579	26540	13280	14071
Shift	12177	41280	37849	15037	14186
Random	20959	50907	51395	24686	36282

Table 3.3 The performance of 20 mappings based on the combination of four different allocation strategies and five partitionings of the mesh Ω_h (used in BENCH1) with respect to satisfiability of criterion (v) as it is measured by the function $\max_{1 \leq i \leq P} \{\sum_{D_j \in N(D_i)} c(D_i, D_j) \times d(m(D_i), m(D_j))\}$.

Algorithms	Hybrid	CM_Clust.	Rec_Bisec.	PxQ	1xQ
$GBA\_d_{1,1}$	454	5417	2399	609	1640
QAP	373	3723	1238	427	687
Shift	529	4326	2690	482	821
Random	731	5095	2663	857	1875

Table 3.4 Execution time (in sec) of four allocation strategies of a 64-way partitioning of the 18890 element triangular mesh on a SUN Sparc 2.

Allocation Algor.	Time
GBA	1.416
QAP	105.616
Shift	0.0166
Random	0.0166

Table 3.5 Total elapse time of the parallel Jacobi-SI for the 20 different mappings on the nCUBE-6400 with 64 processors.

Algorithms	Hybrid	CM_Clust.	Rec_Bisec.	PxQ	1xQ
GBA	0.865	1.021	0.954	0.925	0.867
QAP	0.865	1.021	0.954	0.925	0.867
Shift	0.865	1.021	0.955	0.925	0.867
Random	0.865	1.020	0.955	0.925	0.867

Table 3.6 Percentage of local communication time of the parallel Jacobi-SI under the 20 different mapping strategies on a nCUBE-6400 with 64 processors.

Algorithms	Hybrid	CM_Clust.	Rec_Bisec.	PxQ	1xQ
GBA	10	22	15	11	5
QAP	10	22	15	11	5
Shift	11	22	15	11	5
Random	10	26	15	11	5

Table 3.7 Percentage of overhead (T_{copy}) introduced other than the local and global synchronization time of the parallel Jacobi-SI mapping using 64 processors on nCUBE-6400; this reflects the impact of the length of the interfaces.

Algorithms	Hybrid	CM_Clust.	Rec_Bisec.	PxQ	1xQ
GBA	1.5	1.5	2.5	2.3	5.7
QAP	1.5	1.5	2.5	2.4	5.7
Shift	1.9	1.5	2.5	2.4	5.8
Random	1.5	1.8	2.5	2.4	5.7

3.4 Scheduling Heuristics

The communication mechanism in the nCUBE-6400 consists of the *send* and *receive* operations. In the case of nCUBE the interconnection network is asynchronous circuit switched where the *send* operation is non-blocking and the *receive* operation is a blocking one. The execution time of *send* is a function of the traffic within the network while for the *receive* it depends on the actual and expected times of the message arrivals.

Next, we present some performance data for the send operation on a 16 node nCUBE-6400 and the RxQ partitioning of a triangular mesh of a rectangular domain with 53,992 elements and 27,303 nodal points of the domain of Figure 2.24. Figure 3.7 (plot corresponding to case without scheduling) depicts the performance of the send operation for the R×Q partitioning, with an upper bound of subdomain connectivity equal to 8, and random allocation of the subdomains to processors. Processor 3 sends data to 8 non-neighbor processors, so it is more likely to have its *send* operation blocked by edge-contention. Figure 3.7 (plot corresponding to the case with scheduling) also depicts the performance of the *send* operation for the Ext_R×Q partitioning, with an upper bound of subdomain connectivity equal to 6, using an optimum allocation of the subdomains to processors and 2-Wave static local scheduling. Processors

3, 7, 13, and 15 have the maximum send execution time, since they have to send data to 6 other neighbor processors. The maximum difference between the execution times is around 1,000 cycles. The execution time of the Ext_R×Q partitioning with the optimum allocation, using 2-dimensional gray code and 2-Wave scheduling mechanism, is less expensive than the R×Q partitioning with random allocation by 1,000 cycles.

Figure 3.8 illustrates the performance of the receive operation for two different implementations. The first implementation is a blocking mechanism for a statically defined ordering (compile time ordering plot) of the local messages and the second one is a non-blocking FIFO mechanism using busy-wait primitives like *ntest* and *nread* of nCUBE-6400 or *crecv* on the iPCS/860 and DELTA. The non-blocking FIFO mechanism (run time ordering plot) is less expensive than the statically defined ordering of the local messages by 10,000 cycles.

Summarizing, Figure 3.9 illustrates that a nearly optimal partitioning and allocation with efficient message scheduling and asynchronous implementation of the receive operation can reduce substantially the execution time of the local synchronization (communication) between the processors.

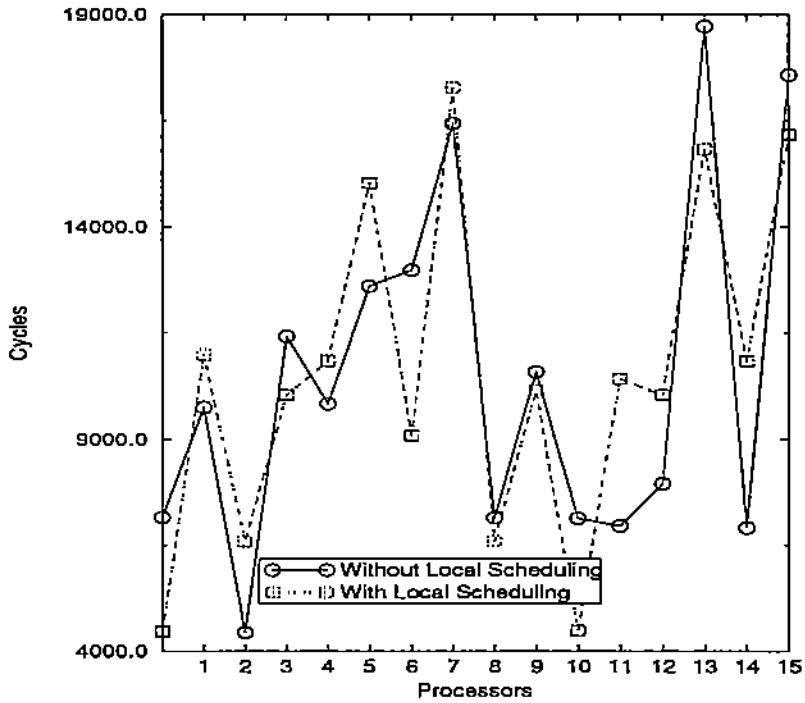


Figure 3.7 Performance of send operation on nCUBE-6400 for a moderate sized problem (close to 27,000 equations).

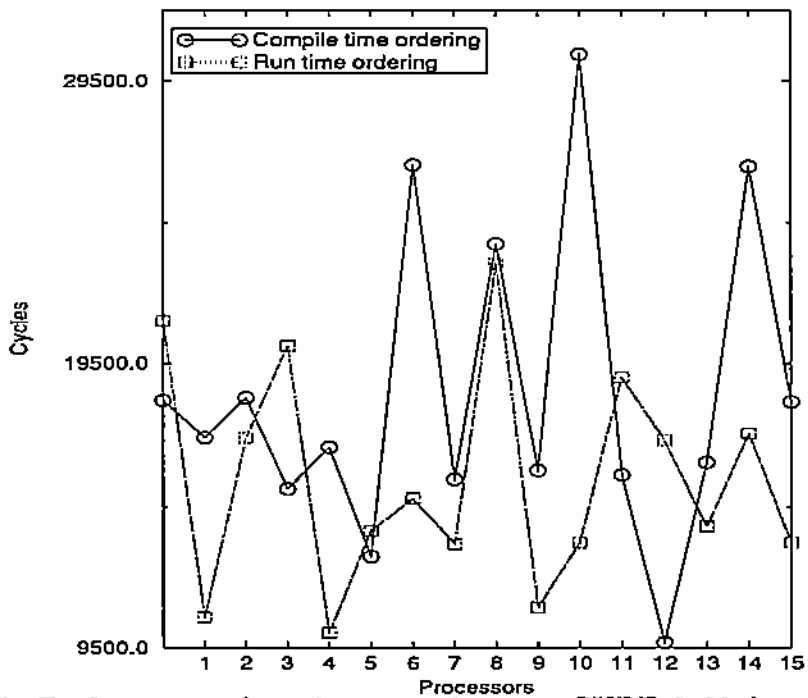


Figure 3.8 Performance of receive operation on nCUBE-6400 for a moderate sized problem (close to 27,000 equations).

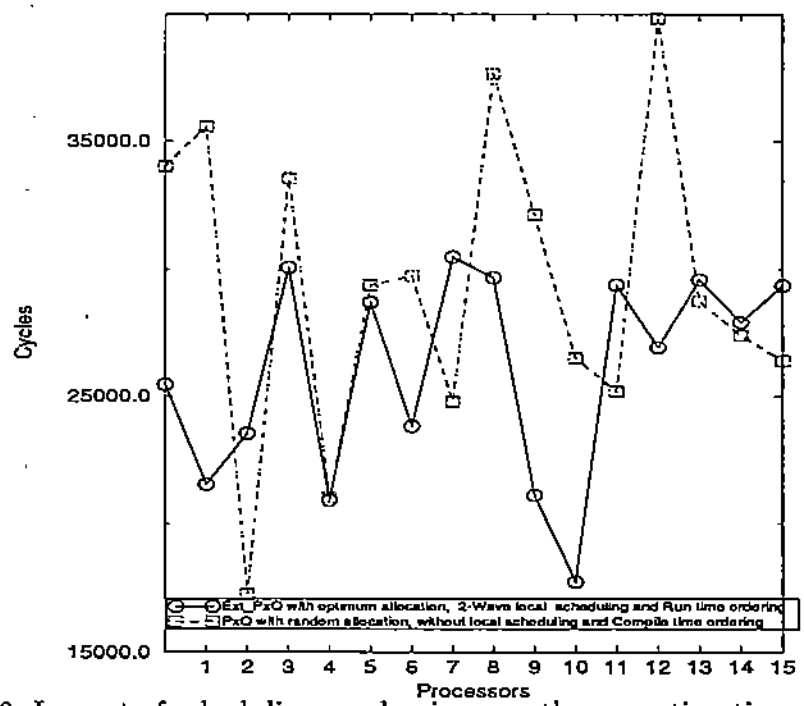


Figure 3.9 Impact of scheduling mechanisms on the execution time (in cycles) of the local communication overhead of the PDE computation.

4. ALGEBRA BASED MAPPING TECHNIQUES

In this chapter we present the parallelization of matrix-vector and matrix-matrix product operations for dense and banded matrices on distributed memory multiprocessor systems that support at least mesh and ring interconnection topology. A new approach is presented for the distribution of the data among the processors which eliminates the synchronization delay and minimizes the communication overhead.

For the case of well structured computations, special purpose algorithm / architecture pairs were suggested, known as systolic arrays (see [Kun82], [Mol82], [MW84]). These architectures consist of simple processing elements (PEs) which are capable of performing one arithmetic operation. In systolic computations, the decrease of edge contention and synchronization is achieved by mapping the computation graph into a systolic array such that the correct data are in the correct place at the appropriate time. This scheduling strategy usually results in minimizing the time any PE spends waiting to receive the required data.

We propose to develop systolic type techniques to design faster algorithms/software for MIMD course grain computation / architecture pairs. To test these approaches we have selected to parallelize some primitive linear algebra operations known as BLAS 2 and 3 kernels [Dong 88, 88a]. For the case of shared and hierarchical memory machines, it has been shown that efficient portable parallel vector code can be designed using these kernels [Dayd 90]. It appears that the parallelization of BLAS for distributed memory machines is less developed. In sections 4.1, 4.2.1, and 4.3.1, we review some of the known matrix multiplication algorithms and their complexity for various architectures. Sections 4.2.2 and 4.3.2 we present the proposed matrix multiplication algorithms for banded matrices. The implementation and performance of

these new algorithms on nCUBE-6400 is discussed in chapter 5. The results indicate almost linear speed-up on a 64 processors configuration nCUBE-6400 with one Mbyte of memory per processor.

4.1 Overview of Parallel Matrix Multiplication Algorithms

Matrix and vector operations are at the core of many important scientific computations. Many problems in physics, mathematics, engineering and chemistry can be formulated as matrix-vector operations. A lot of effort is dedicated in finding an efficient method to multiply matrices with vectors. In this section we present some attempts in this field. These attempts are by no means meant to be complete; it is just a summary for some recent work in this field.

Fox, Otto, and Hey in [GOH87] proposed techniques for matrix multiplication. Their method depends on the partitioning of the matrix into square or rectangular subblocks. These blocks are distributed between the processors. After the completion of the matrix multiplication operation, the product matrix is distributed among the processors in the same fashion. The algorithm depends on exploiting the mesh architecture that can be embedded in any hypercube architecture. It also depends on broadcasting some of these data blocks.

Deckel, Nassimi, and Sahni in [DNS81] proposed a matrix multiplication algorithm for cube connected and perfect shuffle computers. They used N^2m processors to multiply two $N \times N$ matrices in $O(\frac{N}{m} + logm)$ time. They also showed how $m^2 processors, 1 \leq m \leq N$, can be used to multiply two $N \times N$ matrices in $O(\frac{N^2}{m} + m \times (\frac{n}{m})^{2.61})$ time. This method is efficient for multiplying dense matrices, but, it will not be very efficient for matrix vector operations of banded matrices.

Johnsson in [Joh85] presented algorithms for dense matrix multiplication and for Gauss-Jordan and Gaussian elimination. His algorithm can run on any boolean cube or torus computers. His algorithms achieves a 100% processor utilization except for a latency period $T_{latency} = O(n)$ for an n-cube systems.

Berntsen in [Ber89] proposed an efficient algorithm for dense matrix multiplication. Berntsen's idea is to partition the hypercube into a set of subcubes and using the cascaded sum algorithm to add up the contributions to the final matrix. His idea also reduced the asymptotic communication to $\frac{N^2}{P^{\frac{2}{3}}}$ on the expense of having $\frac{N^2}{P^{\frac{2}{3}}}$ extra bytes of memory per processor (P is the number of processors). In [JHM89], Johnsson et al. presented a data parallel matrix multiplication algorithm. Their algorithm was implemented on the Connection Machine CM-2, their implementation has a peak overall performance of 5.8 GFLOPS.

Most of the previous work on this subject either uses broadcasting or it is not efficient for banded-matrix operations. In this thesis, we present four algorithms for matrix-matrix and matrix-vector operations. The four algorithms require communication between neighboring processors and minimize synchronization delays.

4.2 Parallelization of level 2 BLAS operations

In this section we are considering a parallel implementation of level 2 BLAS operations on a wrap around linear array and on a grid of P processors respectively. These operations involve the matrix-vector operation c = β c + α A b, where A is a square matrix, b, c are column vectors of compatible dimensions, and α, β are real scalars.

4.2.1 Dense Matrix × Vector Multiplication

First, we consider the implementation of the operation c = β c + α A b on a linear wrap around array of P processors. We assume that the input data A and b, c are decomposed into the submatrices $A_{i,j} \in R^{\frac{N}{P} \times \frac{N}{P}}$ and the subvectors $c_i, b_i \in R^{\frac{N}{P}}$ respectively. Each processor, i, contains the block row $\{A_{i,j}\}_{j=1}^{P}$ and the subvectors b_i, c_i and computes the updated subvector c_i using the $A_{i,(i+1)modP}$ submatrix. At last it receives the subvector b_i from the $(i+1)modP$ processor.

The interconnection of PEs which is a *Folded Linear Array* and the distribution of input are shown in Figure 4.1. PE i computes the subvector c_i. As it can be seen

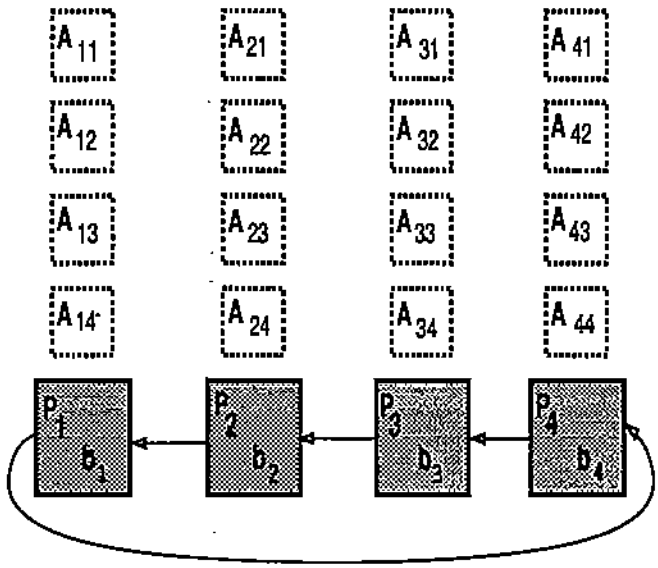


Figure 4.1 The interconnection network and distribution of the matrix by rows.

the matrix A is divided into the submatrices $A_{i,j} \in R^{\frac{N}{P} \times \frac{N}{P}}$, where P is the number of processors, and the vector b is divided into the subvectors $b_i \in R^{\frac{N}{P}}$. Each processor contains one row of submatrices and one subvector.

Algorithm 4.1

The vector c = (c_i) can be expressed as :

$$c_i = \sum_{j=1}^{N} A_{ij} b_j \tag{4.1}$$

The algorithm requires P iterations. In each iteration a partial sum of equation (4.1) is accumulated. The algorithm starts by multiplying $A_{i,i}$ by b_i. Then, every processor sends the part of vector b it has in processor $i-1$ and receives the part of b from the corresponding processor. Finally, it multiplies it by $A_{i,(i+1) mod P}$. The algorithm can be expressed as following :

```
For all PE i = 1 to P do
  begin
    For k = 1 to P  do
      begin
        Send  b(i) to PE  i-1 mod P
        c(i) = c(i) + A(i,i+k mod P) * b(i)
        Receive b(i) from PE i+1 mod P
      end
  end
```

Complexity Analysis

Assume that each multiply and add operation takes τ seconds. Also, we assume that the transferring of w words in a network without edge contention takes $\alpha+\beta\times w$, where $\alpha = \alpha(d)$ and d is the length of the message path. Both α and β are machine dependent parameters. Under the above assumptions the execution time for the algorithm is :

$$T_P = P \times \{\frac{N^2}{P^2} \times \tau + \alpha + \beta \times \frac{N}{P}\} \tag{4.2}$$

Since

$$T_1 = N^2 \times \tau \tag{4.3}$$

we get speedup equal to :

$$S(N,P) = \frac{N^2 \times \tau}{P \times \{\frac{N^2}{P^2} \times \tau + \alpha + \beta \times \frac{N}{P}\}} \tag{4.4}$$

The space required for each processor is : $O(\frac{N^2}{P^2} + 2 \times \frac{N}{P})$

4.2.2 Banded Matrix × Vector Multiplication

In this section we investigate the implementation of $c = A \times b$, where $c, b \in R^N$, and $A \in R^{N\times N}$ is a banded matrix with w_1 being the upper bandwidth and w_2 the lower bandwidth of the matrix A. Throughout this thesis we assume that the matrices are stored using a sparse scheme [Rice85]. For simplicity we assume that $N = P$. Usually in practice $N >> P$. However, the case $N >> P$ can easily be generalized by replacing each element $a_{i,j}$ by a submatrix $A_{i,j} \in R^{\frac{N}{P}\times\frac{N}{P}}$.

The interconnection of PEs which is a *Linear Array* and the distribution of input are shown in Figure 4.2. PE i computes the subvector c_i. As it can be seen the matrix A is split into two submatrices, the strictly lower triangular submatrix of A, let us call it L, and the upper triangular submatrix of A, let us call it U, such that $A = L + U$. Each processor contains one row of elements (in the general case a strip of rows) and one element of vector b (in the general case a strip of rows).

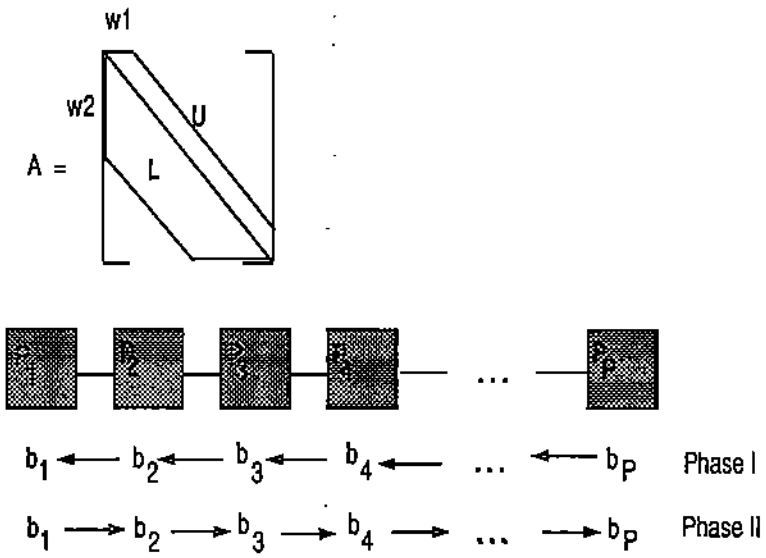


Figure 4.2 The interconnection network and distribution of the input.

Algorithm 4.2

The vector c can be expressed as : $c = L \times b + U \times b$. The algorithm consists of 2 phases. In the first phase it multiplies U by b where a number of $w_1 + 1$ iterations is required. In the second phase it multiplies L by b where a number of w_2 iterations is required. In the first phase each processor i multiplies a_{ii} by b_i and sends b_i to processor $i - 1$ and it also receives the new part of the vector b form processor $i + 1$. Processor $i = 0$ during the sending stage sends nothing, while processor $i = P$ during the receiving stage receives nothing. At the $k^{th}_{=}$ iteration $i > P - k + 1$ processors remain idle. In the second phase each processor restores b_i from temporary storage,

hence processor i contains b_i and sends it to processor $i+1$ and then it multiplies a_{ii-1} by b_{i-1}. Processor $i = P$ during the sending stage sends nothing while processor $i = 0$ during the receiving stage receives nothing. The algorithm can be expressed as following :

Phase 1: Multiply the Upper triangular U by b

```
temp := d
For each PE i do in parallel
  For j := 1 to w2
    if (i + j  =< P) then
       begin
         if ( i = 1 ) then do nothing
         else Send d to PE i-1
         c := c + a(i, j+i) * d
         if ( i = P ) then do nothing
         else Receive d from PE i+1
       end
    endif
  end
end
```

Phase 2: Multiply the Lower triangular L by b

```
For each PE i do in parallel
  begin
  d := temp
    For j := 1 to w2
      if (i < j) then
         begin
           if ( i = P ) then do nothing
           else Send d to PE i + 1
           if ( i = 1 ) then do nothing
           else Receive d from PE i - 1
           c := c + a(i, i-j) * d
         end
      endif
    end
end
```

Complexity Analysis

Without lost of generality we assume that the matrix A has K non-zero elements and $N >> w_1 + w_2 + 1$. Under the above assumptions on the time required to

communicate and multiply/add a datum we get :

$$T_P = \frac{K}{P} \times \tau + (w_1 + w_2 + 1) \times \{\alpha + \beta \times \frac{N}{P}\} \tag{4.5}$$

Since

$$T_1 = K \times \tau \tag{4.6}$$

we get speedup equal to :

$$S(N,P) = \frac{K \times \tau}{\frac{K}{P} \times \tau + (w_1 + w_2 + 1) \times \{\alpha + \beta \times \frac{N}{P}\}} \tag{4.7}$$

The space required for each subdomain is : $O(\frac{K}{P} + 3 \times \frac{N}{P})$

4.3 Parallelization of level 3 BLAS operations

In this section we are considering a parallel implementation of level 3 BLAS operations i.e., the matrix-matrix operation C = β C + α A B where A, B, C are matrices of dimensions M-by-K, K-by-N and M-by-N respectively, and α, β are real scalars. Our current implementation applies only to square matrices (N = M = K).

4.3.1 Dense Matrix × Dense Matrix Multiplication

First, we consider the implementation of the matrix-matrix operation C = α C + β A B on a wrap around grid of P processors. We assume that the input data A, B are decomposed into submatrices $A_{i,j}, B_{i,j} \in R^{\frac{N}{\sqrt{P}} \times \frac{N}{\sqrt{P}}}$ which are stored in each processor (i,j). The product submatrix $C_{i,j}$ is computed in $\sqrt{P}$ iterations. If we suppress the block indices, then the computation carried out by each processor in the kth iteration consists of sending B, C to processors (i, (j-1)mod P) and ((i+1)mod P, j) respectively, computing $C = C + AB$, and receiving B and C from processors (i, (j+1) mod $\sqrt{P}$) and ((i-1) mod $\sqrt{P}$, j) respectively.

The interconnection of PEs which is a *Folded Grid* topology and the distribution of input are shown in Figure 4.3.

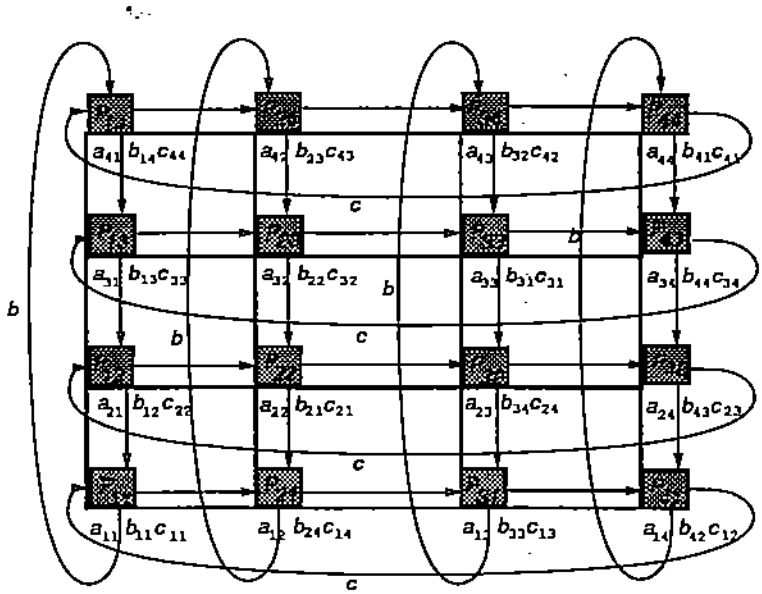


Figure 4.3 The interconnection network and the distribution of input.

PE (i,j) computes the submatrix $C_{i,j}$. As it can be seen the matrices A,B are divided into the submatrices $A_{i,j}, B_{i,j} \in R^{\frac{N}{\sqrt{P}} \times \frac{N}{\sqrt{P}}}$, where P is the number of processors. There are two paths across which data are moved : the c-path across which the algorithm moves the submatrices C_{ij} and the b-path across which it moves the submatrices B_{ij}.

Algorithm 4.3

The matrix $C = (C_{i,j})$ can be expressed as :

$$C_{i,j} = \sum_{k=1}^{N} A_{jk} B_{k,j}$$

For square matrices the interconnection is organized as folded square grid $\sqrt{P} \times \sqrt{P}$. In each processor (i,j) we store the submatrices $A_{i,j}, B_{j,i} \in R^{\frac{N}{\sqrt{P}} \times \frac{N}{\sqrt{P}}}$; initialize $C_{i,j}$ to zero; throughout this section we will refer to them as A, B, C. In processor

(i,j) the submatrix $C_{i,j}$ is computed after $\sqrt{P}$ iterations. Each iteration consists of the following three stages : (1) Send B, C across b-path and c-path accordingly to processors (i, (j-1) mod $\sqrt{P}$), and ((i+1) mod $\sqrt{P}$, j). (2) Compute : $C = C + A \times B$. (3) Receive B,C from processors (i, (j+1) mod $\sqrt{P}$), and ((i-1) mod $\sqrt{P}$, j) accordingly. The algorithm can be expressed as follows :

```
For each PE (i,j) do in parallel
  For iter := 1, sqrt(P) do
    begin
      Send B across b-path to (i, (j-1) mod sqrt(P))
      Send C across c-path to ((i+1) mod sqrt(P), j)
      C := C + A * B
      Receive B from processor (i, (j+1) mod sqrt(P))
      Receive C from processor ((i-1) mod sqrt(P), j)
    end
end
```

Complexity Analysis

Under the same assumptions on the time required to communicate and multiply/add on a datum we get :

$$T_P = \sqrt{P} \times \{\frac{N^3}{P^{\frac{3}{2}}} \times \tau + 2 \times (\alpha + \beta \times \frac{N^2}{P})\} \tag{4.8}$$

Since

$$T_1 = N^3 \times \tau \tag{4.9}$$

we get speedup equal to :

$$S(N,P) = \frac{N^3 \times \tau}{\sqrt{P} \times \{\frac{N^3}{P^{\frac{3}{2}}} \times \tau + 2 \times (\alpha + \beta \times \frac{N^2}{P})\}} \tag{4.10}$$

The space required is : $O(3 \times \frac{N^2}{P^2})$

4.3.2 Banded Matrix × Banded Matrix Multiplication

Second, we consider the implementation of $C = \beta C + \alpha AB$, on a ring of P processors where A, B are banded matrices with u_1, u_2 upper and l_1, l_2 lower bandwidths

respectively. Again we describe the realization for N = P. The case $N >> P$ is a straightforward generalization. The processor i computes column C_i of matrix C and holds one row of matrix A (denoted by A_i) and a column of matrix B (denoted by B_i).

The interconnection of PEs which is a *Linear Array* and the distribution of input are shown in Figure 4.4.

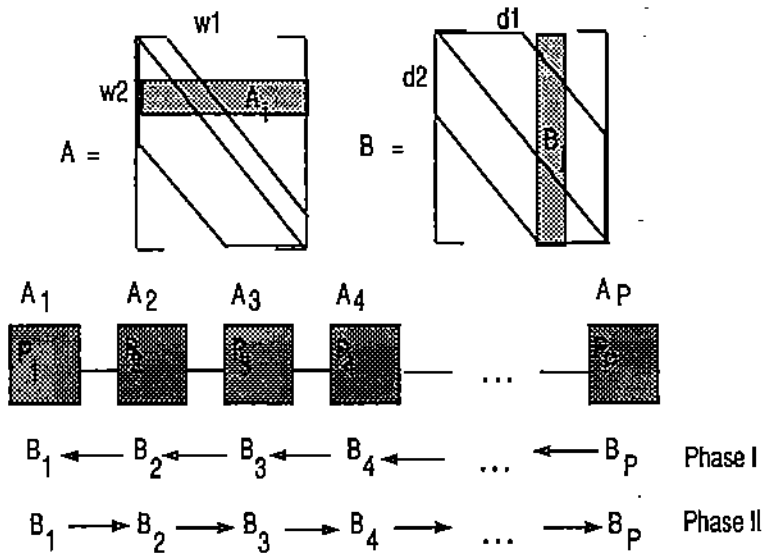


Figure 4.4 The interconnection network and the distribution of input.

PE i holds the column C_i of matrix C. In each processor we store one row of elements (in the general case a strip of rows) from matrix A and one column of elements (in the general case a strip of columns) from matrix B.

Algorithm 4.4

The algorithm consists of two phases as in banded matrix vector multiplication. Without lost of generality we can assume and $\alpha = 1$ and $\beta = 1$. In the first phase, each PE starts by calculating $c_{ii} = c_{ii} + A_i \times B_i$, then each PE i passes B_i to PE $i - 1$. This phase is repeated for $\min\{u_1, u_2\} + 1$ times. In the second phase each

PE restores B_i and passes it to PE $i+1$. This phase is repeated l_i times, where $l_i = \min\{l_1, l_2\}$.

Phase 1

```
temp := b
For each PE i do in parallel  /* each PE contain a = Ai , b = Bi */
  For j := 0 to min(u1 , u2)
    if (i + j =< N) then
       begin
         if (i = 1) do nothing
         else Send b to PE i-1
         c(i,i+j) := c(i,i+j)  +  a * b
         if (i = P) then do nothing
         else Receive b from PE i+1
    endif
  endfor
endfor
```

Phase 2

```
b := temp
For Each PE i in parallel do
  For j := 1 to min(l1 , l2) do
    if (i > j) then
       begin
         if(i = P) then do nothing
         else send b to PE i+1
         if( i = 1) then do nothing
         else receive b from PE i-1
         c(i, i-j) := c(i,i-j) +  a * b
    endif
  endfor
endfor
```

Complexity Analysis

Without lost of generality, we assume that K_1, K_2 are the number of non-zero elements for the matrices A, B respectively and denote by $w_1 = u_1 + l_1 + 1$ and $w_2 = u_2 + l_2 + 1$. Then we can show that

$$T_P = \frac{\min(K_1 w_2, K_2 w_1)}{P}\tau + \{\alpha + \beta\frac{N}{P}\min(w_1, w_2)\}min(w_1, w_2) \qquad (4.11)$$

Since

$$T_1 = \min(K_1 \times w_2, K_2 \times w_1) \times \tau \tag{4.12}$$

we get speedup equal to

$$S(N,P) = \frac{\min(K_1 \times w_2, K_2 \times w_1) \times \tau}{\frac{\min(K_1 w_2, K_2 w_1)}{P}\tau + \{\alpha + \beta\frac{N}{P}\min(w_1, w_2)\} min(w_1, w_2)} \tag{4.13}$$

5. PERFORMANCE EVALUATION OF ALGEBRA BASED MAPPING TECHNIQUES

To verify our theoretical analysis of matrix-vector multiplication and matrix matrix multiplication algorithms, we implemented them on nCUBE-6400. For the implementation we used the fortran language and nwrite, nread communication primitives. To justify the scalability of the algorithms we measured the speedup using three different definitions. For distributed memory multiprocessor systems fixing the size problem creates a constraint since large size data cannot fit on a single processor. In such cases the scaled speedup can be computed either as :

$$Scl\_SpUp1 = \frac{Mflops \;\; using \;\; P \;\; processors}{Mflops \;\; using \;\; single \;\; processor} \tag{5.1}$$

or as :

$$Scl\_SpUp2 = P \times \frac{T_{Work\ done\ by\ P\ proces} - T_{Work\ wouldn't\ done\ by\ serial\ proces}}{T_{Work\ done\ by\ P\ proces}} \tag{5.2}$$

The time (in μsec) to communicate a zero byte, in a network without edge contention is :

$$T_{initial} = 2.84 \times d + 132.87$$

The floating-point performance of the node was measured to be in the range

$$t_{flop} = 1.28\mu sec \rightarrow 1.40\mu sec \tag{5.3}$$

The time to perform the operation c(i) = c(i) + a(i,j) * b(j) was measured (without optimization) to be equal to

$$t_{oper} = 5.554\mu sec \tag{5.4a}$$

The time to perform the operation c(i) = c(i) + a(i,j) * b(j) was measured (with the optimization) to be equal to

$$t_{oper} = 3.852\mu sec \tag{5.4b}$$

The time to perform the operation c(i) = c(i) + a(i, j) * b(id(i,j)) was measured (without optimization) to be equal to

$$t_{oper} = 7.915\mu sec \tag{5.5}$$

The time to perform the operations (1_{oper}) : rsum = rsum + a(i, j) * b(j, k) and (2_{oper}) : c(i,j) = c(i,j) + rsum were measured (without optimization) to be equal to

$$t1_{oper} = 3.933\mu sec \quad and\ t2_{oper} = 7.632E-03 \tag{5.6}$$

The time to perform the operations (3_{oper}) rsum = rsum + a(i, k) * b(id(i,k), j)) was measured (without optimization) to be equal to

$$t3_{oper} = 7.639\mu sec \tag{5.7}$$

5.1 Dense Matrices

Table 5.1 depicts the measured Mflops of the algorithm 4.1 presented in section 4.2.1 which performs the dense matrix vector multiplication using a single processor and also it depicts the Mflops using 64 processors as well as the fixed speedup and scaled speedup computed using the formulas (5.1) and (5.2) for $A \in R^{N \times N}$, where N = 320, 640, 1600.

From the equations (4.4) and (5.1) - (5.6) we estimate the speedup of the algorithm 4.2.1 for the matrix vector multiplication on nCUBE-6400 and for various problem sizes and different configurations. Figure 5.2 depicts this estimated speedup for problem sizes N = 640, 3200, 32000 and processors P = 2^i, for i = 0, 4, 6, 7, and 8.

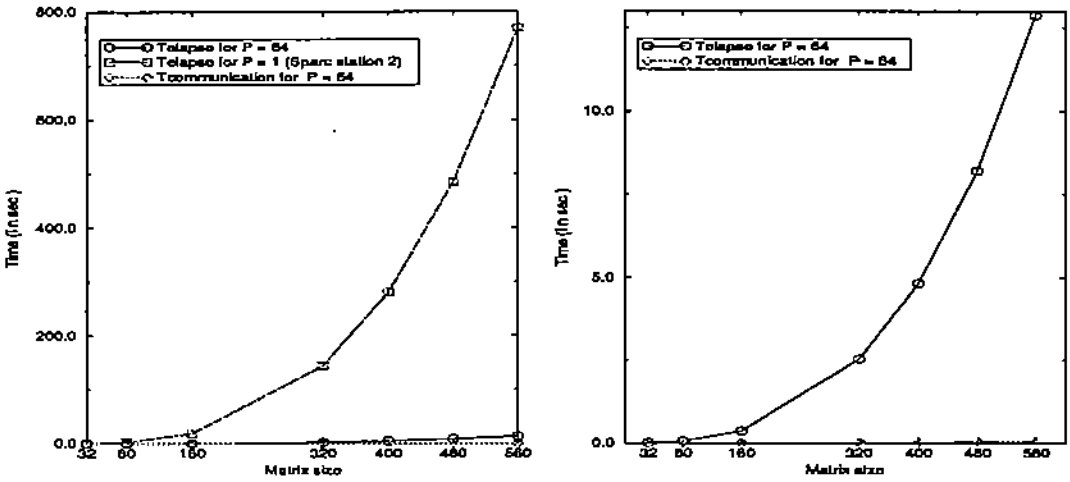


Figure 5.1 Left) Total elapse time of the algorithm 4.1, section 4.2.1, for the dense matrix vector multiplication on nCUBE-6400 and 64 processors, and of the classical algorithm for matrix vector multiplication on Sparc station 2. Right) Total elapse and communication time of the algorithm 4.2.1 on nCUBE-6400 and 64 processors.

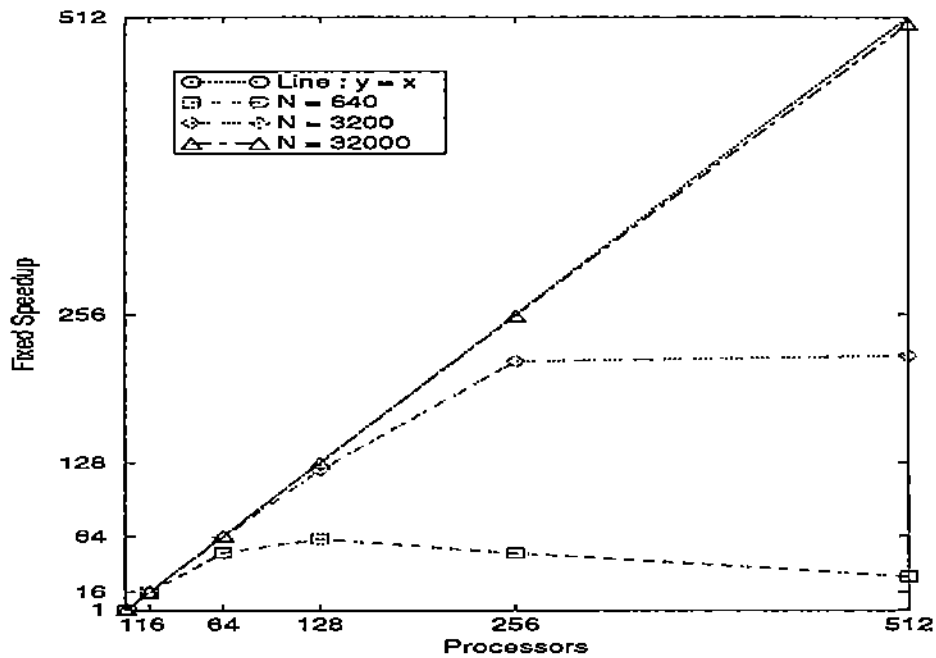


Figure 5.2 Estimated speedup of the algorithm 4.1, section 4.2.1, for the dense matrix vector multiplication on nCUBE-6400 with P = 1, 4, 16, 64, 128, 256, and 512 processors.

Table 5.1 Measured Mflops and the speedup of the algorithm 4.1, section 4.2.1, for the dense matrix vector multiplication on nCUBE-6400 using 64 processors and sizes of the matrix equal to N = 320, 640, 1600.

N	Mflops 1 p	Mflops P p	S(N, P)	Scl_SpUp1	Scl_SpUp2
320	.439	7.359	16.46	16.76	35.12
640	.446	17.048	38.14	38.22	46.63
1600	.447	28.292	-	63.29	59.50

Table 5.2 depicts the measured Mflops of the algorithm 4.3 presented in section 4.3.1 which performs the dense matrix matrix multiplication using a single processor and also depicts the Mflops using 64 processors as well as the fixed speedup and scaled speedup computed using the formulas (5.1) and (5.2) for $A \in R^{N \times N}$, where N = 160, 280, 360, 560.

Table 5.2 Measured Mflops and speedup of the algorithm 4.3 for dense matrix-matrix multiplication on nCUBE-6400 using 64 processors, and sizes of the matrix equal to N = 160, 280, 360, 560.

$\frac{N}{\sqrt{P}} \times \frac{N}{\sqrt{P}}$ / node	Mflops 1 p	Mflops 64 p	S(N, P)	Scl_SpUp1	Scl_SpUp2
20 x 20	0.440	22.870	51.991	51.977	55.150
35 x 35	0.441	25.117	55.966	56.954	58.873
45 x 45	0.441	25.794	58.437	58.489	59.976
70 x 70	0.441	26.664	60.351	60.462	61.373

The difference between the Scl_SpUp1 and Scl_SpUp2 is due to the fact that the parallel algorithm requires more logical operations as well as more evaluation of indices than the serial algorithm. Hence, the Scl_SpUp1 is much smaller than Scl_SpUp2 who reflects the sum of the individual processors apparent efficiencies (compute time

/ total time). From the equations (4.10) and (5.1) - (5.6) we estimate the speedup of the algorithm for dense matrix matrix multiplication on nCUBE-6400 for various problem sizes and different configurations. Figure 5.3 depicts the estimated speedup for problem sizes N = 160, 560, 1200 and processors $P = 2^i$, for i = 0, 4, 6, 7, and 8.

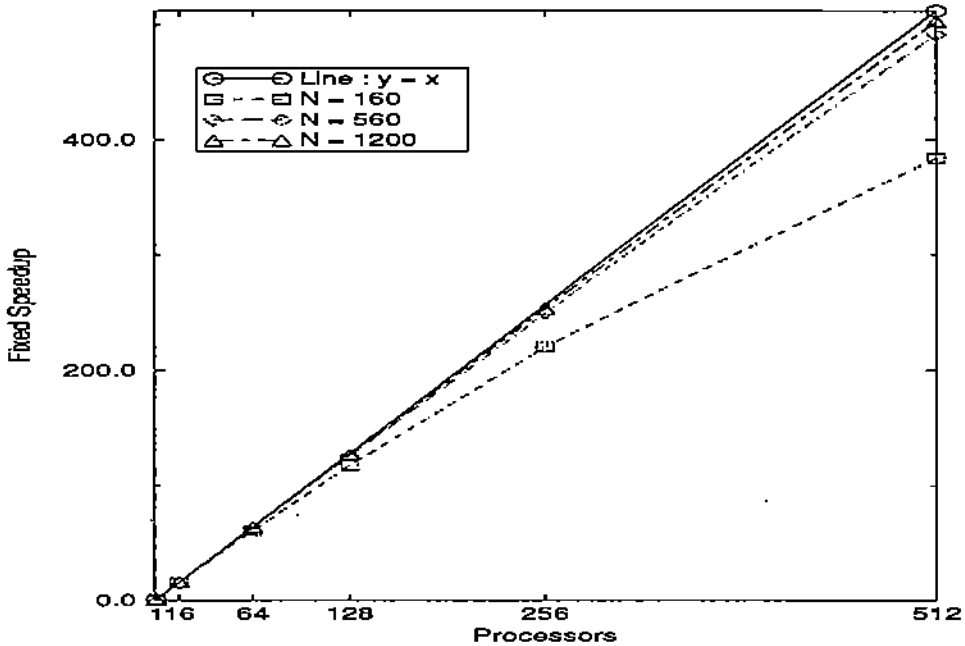


Figure 5.3 Estimated SpeedUp of the algorithm 4.3, section 4.3.1, for the dense matrix matrix multiplication on nCUBE-6400 and P = 1, 4, 16, 64, 128, 256 and 512 processors.

5.2 Banded Matrices

By fixing the size of the matrix for each processor the resulting performance curves should idealy show constant elapse time as a function of the number of processors. Tables 5.3 and 5.4 realize the scalability of the algorithm for banded matrices with upper bandwidth equal to lower bandwidth equal to 8, 16, 32 and 64.

Table 5.3 Measured total elapsed time (in sec) of the banded matrix vector multiplication algorithm 4.2 section 4.2.2, for block tridiagonal matrices. Each block is of size n × n, where n = 8, 16, 32, 64.

matrix size / node	4 nodes	8 nodes	16 nodes	32 nodes	64 nodes
8 x 24	0.0634	0.0644	0.0644	0.0644	0.0645
16 x 48	0.0221	0.0222	0.0222	0.0222	0.0222
32 x 96	0.0847	0.0848	0.0848	0.0849	0.0849
64 x 192	0.3345	0.3346	0.3346	0.3347	0.3347

Table 5.4 Measured scaled speed up (5.2) of the banded matrix vector multiplication algorithm 4.2, section 4.2.2, for block tridiagonal matrices. Each block is of size n × n, where n = 8, 16, 32, 64.

matrix size / node	4 nodes	8 nodes	16 nodes	32 nodes	64 nodes
8 x 24	3.55	7.03	14.05	28.22	56.25
16 x 48	3.87	7.72	15.43	30.94	61.80
32 x 96	3.96	7.91	15.83	31.66	63.30
64 x 192	3.99	7.97	15.95	31.90	63.80

Table 5.5 Measured total elapsed time (in sec) of the banded matrix - matrix multiplication algorithm 4.4, section 4.3.2, for block tridiagonal matrices. Each block is of size n × n, where n = 8, 16.

matrix size / node	4 nodes	8 nodes	16 nodes	32 nodes	64 nodes
8 x 24	0.275	0.281	0.281	0.281	0.281
16 x 48	4.029	4.030	4.030	4.030	4.030

Table 5.6 Measured scaled speedup (5.2) of the banded matrix matrix multiplication algorithm 4.4, section 4.3.2, for block tridiagonal matrices. Each block is of size n $\times$ n, where n = 8, 16.

matrix size / node	4 nodes	8 nodes	16 nodes	32 nodes	64 nodes
8 x 24	2.78	7.31	14.62	29.24	58.50
16 x 48	2.76	7.28	14.56	29.12	58.25

6. A SOFTWARE ENVIRONMENT FOR IMPLEMENTING AND VISUALIZING GEOMETRY BASED MAPPING TECHNIQUES

In this chapter we describe a software infrastructure consisting of "fast" heuristics for determining "optimal" mapping of PDE data suitable for domain decomposition methods. Furthermore we describe a software system which assists the user to visualize and manipulate such mappings in the environment of Parallel-ELLPACK system [HRC+90].

6.1 Parallel (//) ELLPACK

In this section we briefly describe the // ELLPACK - an intelligent parallel programming environment for solving PDEs defined on two and three dimensional domains onto MIMD parallel machines. The (//) ELLPACK is a prototype whose objective is : the development of an environment and a methodology for easily mapping and evaluating parallel numerical algorithms for PDEs onto MIMD parallel machines and analyzing the performance of parallel MIMD architectures for large scale scientific computing.

The // ELLPACK, [HRC+90], is implemented on a hardware facility consisting of a graphics workstation supporting the X11 window system and connected to the NCUBE machine through a local network. The software infrastructure includes i) a PDE problem oriented language processor, ii) a geometry processing tool which is capable of generating fixed meshes and their decompositions automatically and interactively, and iii) an algorithm mapper facility for partitioning and mapping the underlying PDE computation.

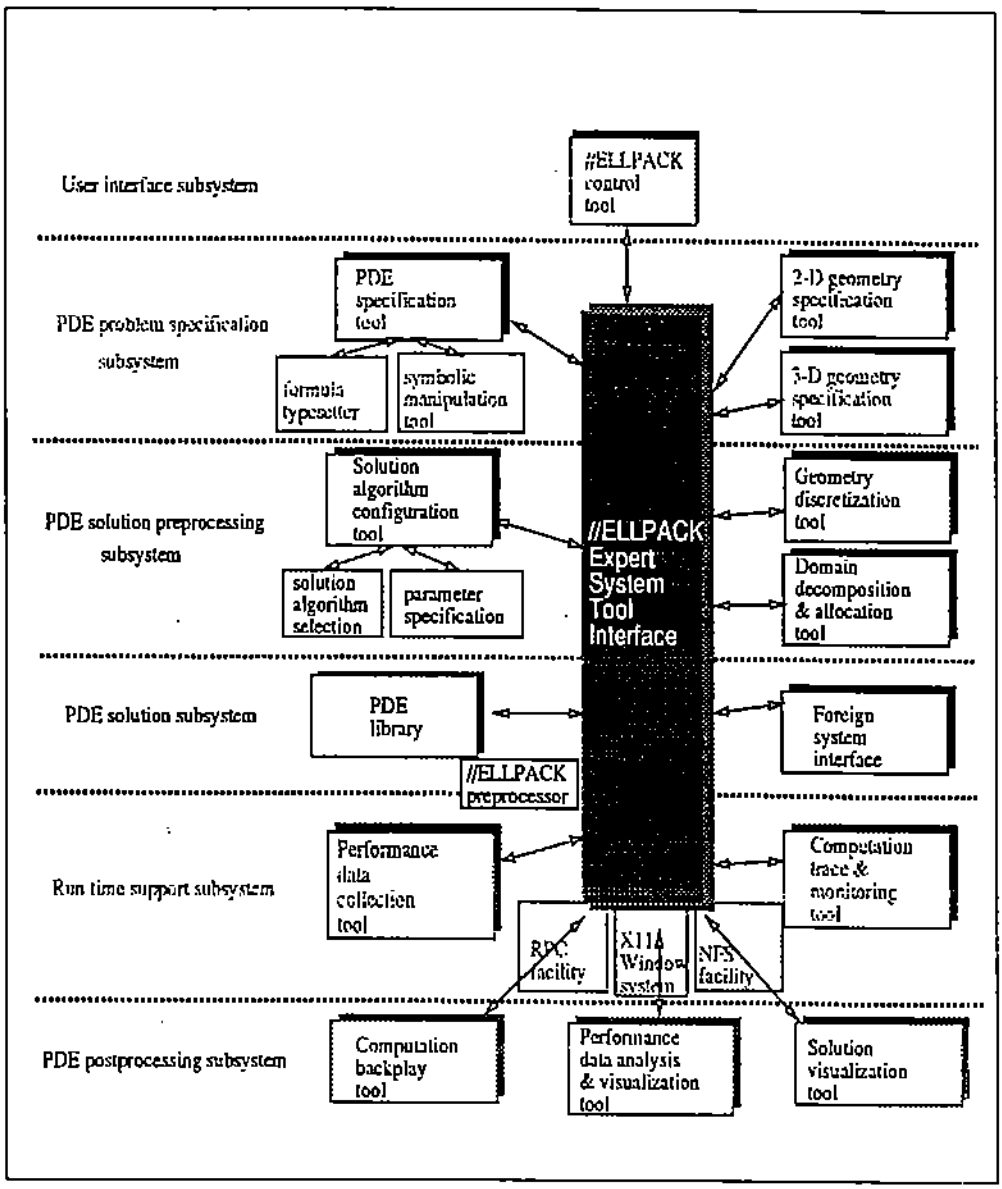


Figure 6.1 The Parallel Ellpack architecture and its hierarchy of editors.

6.2 A software tool for mapping PDE computations to parallel machines : Domain Decomposition Tool

We have built an interactive environment called *DecTool* (short for Domain Decomposer Tool) to help with domain decomposition. An example display is shown in Figure 6.2. DecTool provides facilities for both automatic (using predefined algorithms) and manual decomposition of a given 2-D or 3-D discrete domain. This interactive environment is written using the 5th release of the X11 toolkit known as ATHENA Widgets. DecTool consists of three different windows. The first one is the basic DecTool window which controls the domain decomposition process. This window is shown in the upper left corner of Figure 6.2. Control is implemented through a set of four buttons.

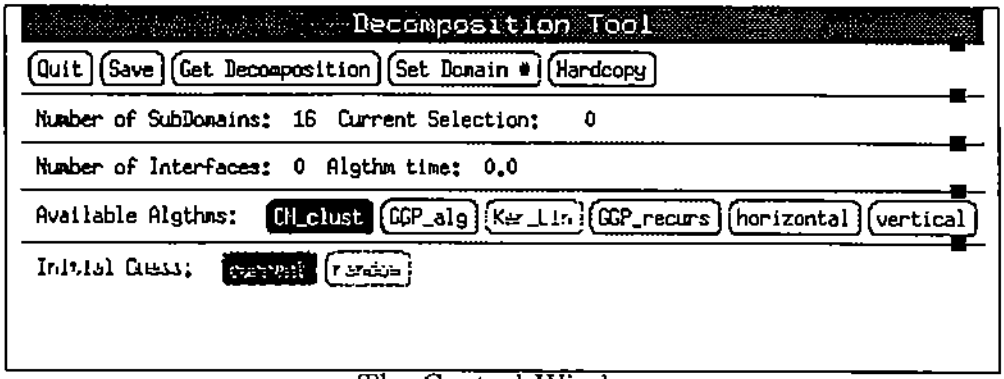
QUIT: Signals to exit from the tool and return to parallel ELLPACK environment (Fig. 5.2).

SAVE: An output file is produced which contains the description of the last decomposition of the domain.

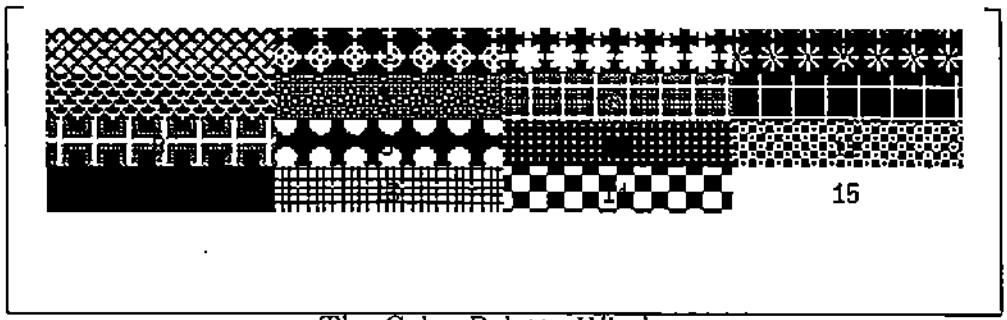
AUTOMATIC: Invokes a specified automatic decomposition algorithm from a library of available algorithms.

SET DOMAIN #: Invokes a dialog window in which the user specifies the number of subdomains (processors).

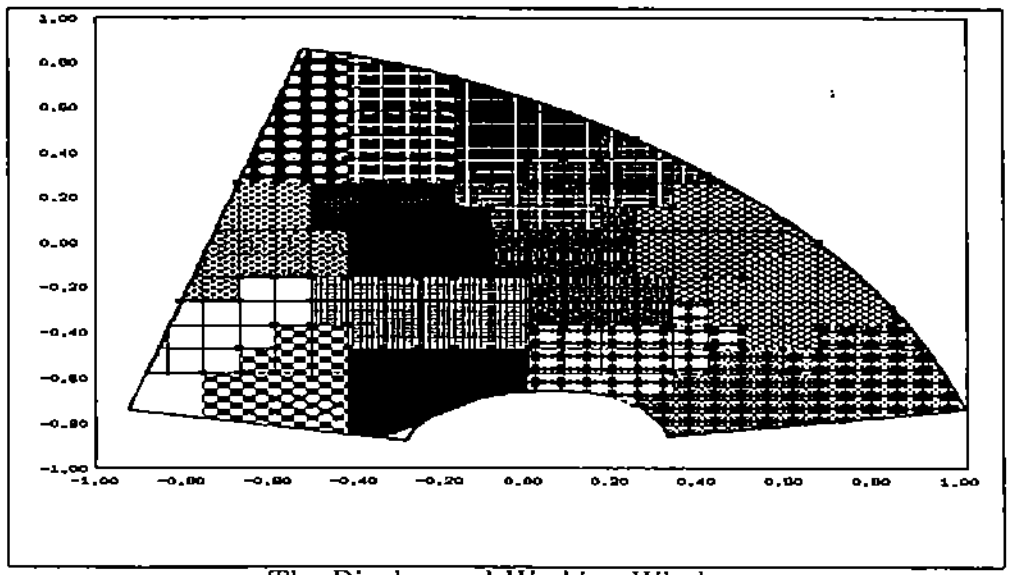
HARDCOPY #: Sends a screen-dump to a laser printer.



The Control Window



The Color Palette Window



The Display and Working Window

Figure 6.2 An instance of the domain decomposition editor consisting of (i) a control window of partitioning heuristics, (ii) the color (or pattern) palette window, and (iii) the display of domain decomposition window.

BIBLIOGRAPHY

BIBLIOGRAPHY

[ANN90] M. Al-Narsa and D. T. Nguyen. An algorithm for domain decomposition in finite element analysis. *Computers and Structures*, 39(3/4):277 – 289, 1990.

[Ber89] J. Berntsen. Communication efficient matrix multiplication on hypercubes. *Parallel Computing*, 12:335–342, 1989.

[Bok81] Shahid H. Bokhari. On the mapping problem. *IEEE Transactions on Computers*, (3):207 – 213, 1981.

[Bok90] Shahid Bokhari. Communication overhead on the Intel iPSC-860 hypercube. Technical Report NAS1-18605, NASA, 1990.

[CAHH92] N. Chrisochoides, M. Aboelaze, E. N. Houstis, and C. E. Houstis. Parallelization of level 2 and 3 BLAS operations on distributed memory machines. In *Proceedings of the Seventh IMACS International Conference on Computer Methods for Partial Differential Equations*, page to appear, 1992.

[CHENH+91] N. P. Chrisochoides, C. E. Houstis, P. N. Papachiou E. N. Houstis, S. K. Kortesis, and J. R. Rice. Domain decomposer: A software tool for mapping PDE computations to parallel architectures. In R. Glowinski et al., editors, *Domain Decomposition Methods for Partial Differential Equations IV*, pages 341–357. SIAM Publications, 1991.

[CHENHR89] N. P. Chrisochoides, C. E. Houstis, S. K. Kortesis E. N. Houstis, and J. R. Rice. Automatic load balanced partitioning strategies for PDE computations. In E. N. Houstis and D. Gannon, editors, *Proceedings of Supercomputing '89*, pages 99–107. ACM Press, 1989.

[CHH91] N. P. Chrisochoides, E. N. Houstis, and C. E. Houstis. Geometry based mapping strategies for PDE computation. In E. N. Houstis and Y. Muraoka, editors, *Proceedings of Supercomputing '91*, pages 115–127. ACM Press, 1991.

[CHK+92] N. P. Chrisochoides, E.N. Houstis, S.B. Kim, M.K. Samartzis, and J.R. Rice. Parallel iterative methods. Technical Report CSD-TR-92-035 and CER-92-16, Department of Computer Sciences, Purdue University, 1992.

[CR87] Tony F. Chan and Diana C. Resasco. A domain-decomposed fast poisson solver on a rectangle. In C. W. Gear R. G. Voigt, editor, *Selected Papers from the Second Conference on Parallel Processing for Scientific Computing*, pages 14–26, Philadelphia, 1987. SIAM.

[CR92] N. P. Chrisochoides and J. R. Rice. Partitioning heuristics for pde computations based on parallel hardware and geometry characteristics. In R. Vichnevetsky et. al, editor, *Proceedings of Seventh IMACS International Conference on Computer Methods for Partial Differential Equations*, page to appear, 1992.

[CSS86] Tony F. Chan, Youcef Saad, and Martin H. Schultz. Solving elliptic partial differential equations on hypercubes. In Michael T. Heath, editor, *Hypercube Multiprocessors 1986*, pages 196–210, Philadelphia, PA, 1986. SIAM.

[CT80] G. Carpaneto and P. Toth. Solution of the assignment problem [h]. *ACM Transactions on Database Systems*, 6:104–111, 1980.

[CT88] T.J. Chan and R. S. Tuminaro. A survey of parallel multigrid algorithms. *Parallel Computers and Their Impact on Mechanics*, 86:155–170, 1988. (A. K. Noor, ed.), AMD.

[DNS81] E. Dekel, D. Nassimi, and S. Sahni. Parallel matrix and graph algorithms. *SIAM Computing*, pages 657–675, 1981.

[Far88] C. Farhat. A simple and efficient automatic fem domain decomposer. *Computers and Structures*, 28:579–602, 1988.

[Far89] C. Farhat. On the mapping of massively parallel processors onto finite element graphs. *Computers and Structures*, 32:347–353, 1989.

[FJL88] G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon and D. Walker *Solving problems on concurrent processors.* Prentice Hall, New Jersey, 1988.

[FM82] Charles M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of the 19th IEEE Design Automation Conference*, pages 175–181, 1982.

[FOS88] J. Flower, S. Otto, and M. Salana. Optimal mapping of irregular finite element domains to parallel processors. *Parallel Computers and Their Impact on Mechanics*, 86:239–250, 1988. (A. K. Noor, ed.).

[Fox86] G. C. Fox. A graphical approach to load balancing and sparse matrix vector multiplication on the hypercube. In M. Schultz, editor, *Proceedings of IMA Institute*, pages 37–51. Springer–Verlag, 1986.

[FYK87] Kunio Fukunaga, Shoichiro Yamada, and Tamotsu Kasai. Assignment of job modules onto array processors. *IEEE Transactions on Circuits and Systems*, C–36(7):888–891, 1987.

[FYSK84] Kunio Fukunaga, Shoichiro Yamada, Harold S. Stone, and Tamotsu Kasai. A presentation of hypergraphs in the euclidean space. *IEEE Transactions on Circuits and Systems*, C–33(4):364–366, 1984.

[Geo73] Alan George. Nested dissection of a regular finite element meshe. *SIAM Journal Numerical Analysis*, 10(2):345 – 363, 1973.

[GGMP88] R. Glowinski, Gene Golub, G Meurant, and J Periaux. *First International Symbosium on Domain Decomposition Methods for PArtial Differential Equations.* SAIM, Philadelphia, 1988.

[Gil80] John Russel Gilbert. *Graph Seperation Theorems and Sparce Gaussian Elimination.* PhD thesis, Stanford University, Department of Computer Science, 1980. STAN-CS-80-833.

[GL78] Alan George and Joseph W. Liu. An automatic nested dissection algorithm for irregular finite element problems. *SIAM Journal Numerical Analysis*, 15(5):1053–1069, 1978.

[GL81] Alan George and J. W. Liu. *Computer solution of large sparce positive definite systems.* Prentice-Hall, Englewood Cliffs, NJ, 1981.

[GM78] Alan George and David R. McItnyre. On the application of the minimum degree algorithm to finite element systems. *SIAM Journal Numerical Analysis*, 15(1):1053–1069, 1978.

[GOH87] G.C.Fox, S. W. Otto, and A.J. Hey. Matrix algorithms on a hypercube I : Matrix multiplication. *Parallel Computing*, pages 17–31, 1987.

[Got81] Satoshi Goto. An efficient algorithm for the two-dimensional placement problem in electrical circuit layout. *IEEE Transactions on Circuits and Systems*, CAS–28:12–18, 1981.

[HRC+90] E. N. Houstis, J. R. Rice, N. P. Chrisochoides, H. C. Karathanasis, P. N. Papachiou, M. K. Samartzis, E. A. Vavalis, Ko-Yang Wang, and S. Weerawarana. //ELLPACK: A numerical simulation programming environment for parallel mimd machines. In J. Sopka, editor, *Proceedings of Supercomputing '90*, pages 97–107. ACM Press, 1990.

[HY81] L.A. Hageman and D.M. Young. *Applied Iterative Methods.* New York, 1981.

[iPS90] int$_e$l Corporation, 3065 Bowers Avenue, Santa Clara, California, 95051,. *iPSC/2 and iPSC/860 User's Guide*, 1990.

[JHM89] S. L. Johnsson, T. Harris, and Kapil K. Mathur. Matrix multiplication on the connection machine. In *Proc. Supercomputing 89, Reno Nevada, ACM Press*, pages 326–332, 1989.

[Joh85] S. L. Johnsson. Communication efficient basic linear algebra computations on hypercube architecture. Technical Report CSD/RR-361, YALE-CS, 1985.

[KG87] David E. Keyes and William D. Gropp. A comparison of domain decomposition techniques for elliptic partial differential equations and their parallel implementation. In C. W. Gear R. G. Voigt, editor, *Selected Papers from the Second Conference on Parallel Processing for Scientific Computing*, pages s166–s202, Philadelphia, 1987. SIAM.

[KL70] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, pages 291 – 307, 1970.

[KRYG82] D.R. Kincaid, J.R. Respess, D.M. Young, and R.G. Grimes. ITPACK 2C: a fortran package for solving large sparse linear systems by adaptive accelerated iterative methods. *ACM Transactions on Mathematical Software*, 6:302–322, 1982.

[Kun82] H. T. Kung. Why systolic architecture. *Computer*, 15:37–46, 1982.

[LF90] M. Loriot and L Fezoui. mesh-splitting preprocessor. unpublished, 1990.

[Liu89a] Josehp W. H. Liu. A graph partitioning algorithm by node seperators. *ACM Transactions on Mathematical Software*, 15(3):198 – 219, 1989.

[Liu89b] Josehp W. H. Liu. The minimum degree ordering with constrains. *SIAM J. Sci. Stat. Comput.*, 10(6):1136 – 1145, 1989.

[LS76] Wai-Hung Liu and Andrew H. Sherman. Comparative analysis for the Cuthill-McKee and the reverse Cuthill-McKee ordering algorithms for sparse matrices. *SIAM Journal Numerical Analysis*, 13(2):199 – 213, 1976.

[Man92] Nashat Mansour. *Physical OPtimization Algorithms for Mapping Data to Distributed-Memory Multiprocessors.* PhD thesis, Computer Science Department, Syracuse University, 1992.

[MO87] R. Morrison and S. Otto. The scattered decomposition for finite elements. *Journal of Scientific Computing*, 2:59–76, 1987.

[Mol82] D. I. Moldova. On the analysis ans synthesis of vlsi algorithms. *Trans. Computers*, pàges 1121–1126, 1982.

[MW84] W. L. Miranker and A. Winkler. Space-time representations of computational structures. *Computing*, 32:93–114, 1984.

[nCU91] nCUBE Corporation, 919 East Hillsdale Boulevard, Foster City, California, 94404. *nCUBE 2 Supercomputers*, 1991.

[PAF90] C. Pommerell, M. Annaratone, and W. Fichtner. A set of new mapping and coloring heuristics for distributed-memory parallel processors. In T. M. Manteuffel, editor, *Proceedings of Copper Mountain Conference on Iterative Methods*, volume 4, pages 1–27, 1990.

[PK89] C.-H. Lee C.-I. Park and M. Kim. Efficient algorithm for graph-partitioning problem using a problem transformation method. *Computer-Aided Design*, 21(10):611 – 618, 1989.

[PS82] Christos H. Papadimitriou and Kenneth Steiglitz. *Compinatorial optimization algorithms and complexity.* Prentice-Hall, Englewood Cliffs, NJ 07632, 1982.

[SE87a] P. Sadayappan and F. Ercal. Cluster-partitioning approaches to mapping parallel programs onto a hypercube. In E. N. Houstis, T. S. Papatheodorou, and C. Polychronopoulos, editors, *Proceedings of Supercomputing '87*, pages 476–497. Springer–Verlag, 1987.

[SE87b] P. Sadayappan and F. Ercal. Nearest-neighbor mappings of finite element graphs onto processor meshes. *IEEE Transactions on Computers*, 36:1408–1420, 1987.

[SS88] Y. Saad and M. H. Schultz. Topological properties of hypercubes. *IEEE Transactions on Computers*, 37:867–871, 1988.

[Swa92] L. Tao Y.C. Zhao K. Thulasiraman M.N.S. Swany. Simmulated annealing and tabu search algorithms for multi-way graph partitioning. Unpublished manuscript, 1992.

[Var62] R.S. Varga. *Matrix Iterative Analysis.* Prentice Hall, New Jersey, 1962.

[Wes83] David H. West. Approximate solution of the quadratic assignment problem. *ACM Transactions on Mathematical Software*, 9:461–466, 1983.

[Wil90] R. D. Williams. Performance of dynamic load balancing algorithms for unstructured mesh calculations. Concurrent Supercomputing Facility, Cal. Tech., 1990.

[WM85] Joe F. Thompson Z. U. A. Warsi and C. Wayne Mastin. *Numerical Grid generation.* North-Kolland, New York, 1985.

VITA

VITA

Chrisochoides Nikos was born on May 4, 1962 in Didimiticho, Greece.